'68' MICRO JOURNAL

$2.95USA

Australia	A $ 4.75	New Zealand	NZ $ 4.75
Singapore	S $ 9.45	Hong Kong	H $23.50
Malaysia	M $ 9.45	Sweden	30:-SEK

VOLUME V ISSUE IV • Devoted to the 68XX User • April 1983

"Small Computers Doing Big Things"

YOUR CHOICE-smart either way

- Over 140 software driven functions
- 82 x 24 or 82 x 20 screen format — software selectable
- High resolution 7 x 12 matrix characters — P-31 green phosphor
- Upper/lower case character set — plus graphics character set
- 56-key alphanumeric keyboard — plus 12-key cursor, numeric pad
- Internal editing functions — insert, delete, scroll, roll, slide, etc.
- Parallel printer I/O port
- 50 to 38,400 baud operation — programmable
- Cursor type, cursor position, print control characters, protected fields, shift inversion, dual intensity and many other features

8212 — twelve-inch diagnonal screen or 8209 — nine-inch diagnonal screen

SWTPC

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241

The UniFLEX™ Operating System extracts every last drop

from the 8 bit 6809 microprocessor allowing it to outperform many 16 bit systems

With the UniFLEX™ Operating System, the 8 bit 6809 microprocessor can perform as well as larger CPUs in a multi-user, multi-tasking environment.

Independently developed from the ground up, UniFLEX™ closely models the features found in the UNIX™ Operating System. And in two years of use, UniFLEX™ has proven the abilities of the 6809 to perform large system functions when incorporated into a properly designed mainframe.

Some of the features supported include:

- full multi-user, multi-tasking capabilities
- hierarchical file systems
- device independent I/O
- four Gigabyte disk capacities
- full file protection
- inter-task communication via pipes
- I/O redirection
- task swapping for efficient memory usage
- full random-access files
- comprehensive shell command language
- foreground-background jobs
- electronic mail and printer spooling
- system accounting facilities

The support software currently available for use under UniFLEX™ is extensive. A sampling of the programs available includes:

- native C compiler (full implementation)
- native Pascal compiler
- FORTRAN 77 ANSI Subset compiler
- COBOL compiler with ISAM files, Report Writer & Sort/Merge
- Extended BASIC interpreter
- Extended BASIC precompiler
- text editing and processing software
- enhanced printer spooler
- variety of absolute and relocatable assemblers
- debug and diagnostic packages

Technical Systems Consultants, Inc. also offers a line of single user FLEX™ software products for 6800 and 6809 processors. For those having an absolute need for a 16 bit processor, UniFLEX™ will be available through OEM licensing arrangements for the 68000 microprocessor. Please call or write for additional information on individual products or OEM licensing arrangements.

UNIX™ is a trademark of Bell Laboratories.
FLEX™ and UniFLEX™ are trademarks of Technical Systems Consultants, Inc.

technical systems consultants, inc.

111 Providence Road
Chapel Hill, North Carolina 27514
(919) 493-1451

'68' MICRO JOURNAL

Portions of the text for 68 MICRO JOURNAL was prepared using the following hard/software.

COMPUTERS-HARDWARE
Southwest Technical Products
219 W. Rhapsody
San Antonio, Texas 78216
S09-5/8 DMF disk-CDS1-8212W-Sprint printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe-DS9-FLEX-Assorted hardware

EDITORS-WORD PROCESSORS
Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX-Editor-Processor

Great Plains Computer Company, Inc.
PO Box 916
Idaho Falls, ID 83401
STYLO-Mail Merge

CONTENTS

Send All Correspondence To:

Computer Publishing Center
68 MICRO JOURNAL
5900 Cassandra Smith
PO Box 849
Hixson, TN 37343
615 842-4600

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, Tenn. and additional entries. Postmaster: send Form 3579 to 68' Micro Journal, PO Box 849, Hixson, Tennessee.

SUBSCRIPTION RATES

USA

1-Year $24.50 2-Years $42.50 3-Years $64.50

FOREIGN

See Page 52

Items Submitted for Publication

Articles submitted for publication should be accompanied by the authors **full name, address, date and telephone number.** It is preferred that articles be submitted on either 5 or 8 inch diskette in TSC Editor format or STYLO format. All diskettes will be returned.

The following TSC Text Processor commands **ONLY** should be used (due to our proportional processor): .sp space, .pp paragraph, .fi fill and .nf no fill. Also please do not format within the text with multiple spaces. The rest we will enter at time of editing.

STYLO commands are all acceptable except the .pg page command, we print edited text files in continous text.

All articles submitted on diskettes should be in TSC FLEX™ format, either FLEX2 6800, or FLEX9 6809 any version.

If articles are submitted on paper they should be on white 8X11 bond or better grade paper. No hand written articles (hand written or drawn art accepted). All paper submitted articles will be photo reproduced. This requires that they be typed or produced with a dark ribbon (no blue), single spaced and type font no smaller than 'elite' or 12 pitch. Typed text should be approximately 7 inches wide (will be reduced to column width of 3 1/2 inches). **Please use a dark ribbon!**

All letters to the editor should also comply with the above and bear a signature. Letters of 'gripes' as well as 'praise' are solicited. We attempt to publish all letters to the editor verbatim, however, we reserve the right to reject any submission for lack of 'good taste'. We reserve the right to define what constitutes 'good taste'.

Advertising: Commercial advertisers please contact the 68 Micro Journal advertising department for current rate sheet and requirements.

Classified: All classified must be non-commercial. Maximum 20 words per classified ad. Those consisting of more than 20 words should be figured at .35 cents per word. 20 words or less $7.50 minimum, one time, paid in advance. No classified ads accepted by telephone.

2MHZ 6809 SYSTEMS

GIMIX offers you a variety to choose from!

38 MB WINCHESTER SYSTEM $17,498.99

HARDWARE FEATURES:

- ★ 2MHz 6809 CPU
- ★ 512KB Static RAM
- ★ 8 RS232C Serial Ports
- ★ 2 Parallel Ports
- ★ DMA Double Density Floppy Disk Controller
- ★ Dual 8" DSDD Floppy Disk System
- ★ Dual Winchester Subsystem with Two19 MB 5¼" Winchester Drives

SOFTWARE FEATURES:

- ★ OS-9 LEVEL TWO Multi-User Operating System
- ★ OS-9 Debugger
- ★ OS-9 Text Editor
- ★ OS-9 Assembler

19 MB WINCHESTER SYSTEM $8998.09

HARDWARE FEATURES:

- ★ 128K Static Ram
- ★ 2MHz 6809 CPU
- ★ 19 MB 5¼" Winchester DMA Subsystem
- ★ 4 RS232C Serial Ports
- ★ 1 MB 5¼" Floppy Disk Drive
- ★ DMA Double Density Floppy Disk Controller

SOFTWARE FEATURES:

- ★ OS-9 LEVEL TWO Multi-User Operating System
- ★ OS-9 Text Editor
- ★ OS-9 Debugger
- ★ OS-9 Assembler

128KB MULTI-USER SYSTEM $6997.39

HARDWARE FEATURES:

- ★ 2MHz 6809 CPU
- ★ DMA Double Density Floppy Disk Controller
- ★ 128KB Static Ram
- ★ 2 RS232C Serial Ports
- ★ Dual 8" DSDD Floppy Disk System

SOFTWARE FEATURES: Your choice of either UniFLEX or OS-9 LEVEL TWO. Both are Unix-like Multi-User/Multi-Tasking Operating Systems.

56KB FLEX / OS-9 "SWITCHING" SYSTEM $4148.49

HARDWARE FEATURES:

- ★ 2MHz 6809 CPU
- ★ 56K Static Ram
- ★ 2 RS232C Serial Ports
- ★ DMA Double Density Floppy Disk Controller
- ★ 2 Built-in 5¼" 40tr DSDD Disk Drives (80 Track DSDD Drive Option . . add $400.00)

SOFTWARE FEATURES:

- ★ GMXBUG monitor — FLEX Disk Operating System
- ★ OS-9 LEVEL ONE Multi-tasking operating system for up to 56K of memory

WINCHESTER SUBSYSTEMS

Winchester packages are available for upgrading current **GIMIX** 6809 systems equipped with DMA controllers, at least one floppy disk drive, and running FLEX, OS-9 LEVEL ONE or OS-9 LEVEL TWO. The packages include one or two 19MB (unformatted) Winchester drives, DMA Hard Disk Interface, and the appropriate software drivers. The Interface can handle two 5¼" Winchester Drives, providing Automatic Data Error Detection and Correction: up to 22 bit burst error detection and 11 bit burst error correction. **UniFLEX NOW AVAILABLE**

Dual drives can be used together to provide over 30 MBytes of on line storage -- or use one for back-up of the other. (More convenient and reliable than tape backup systems.

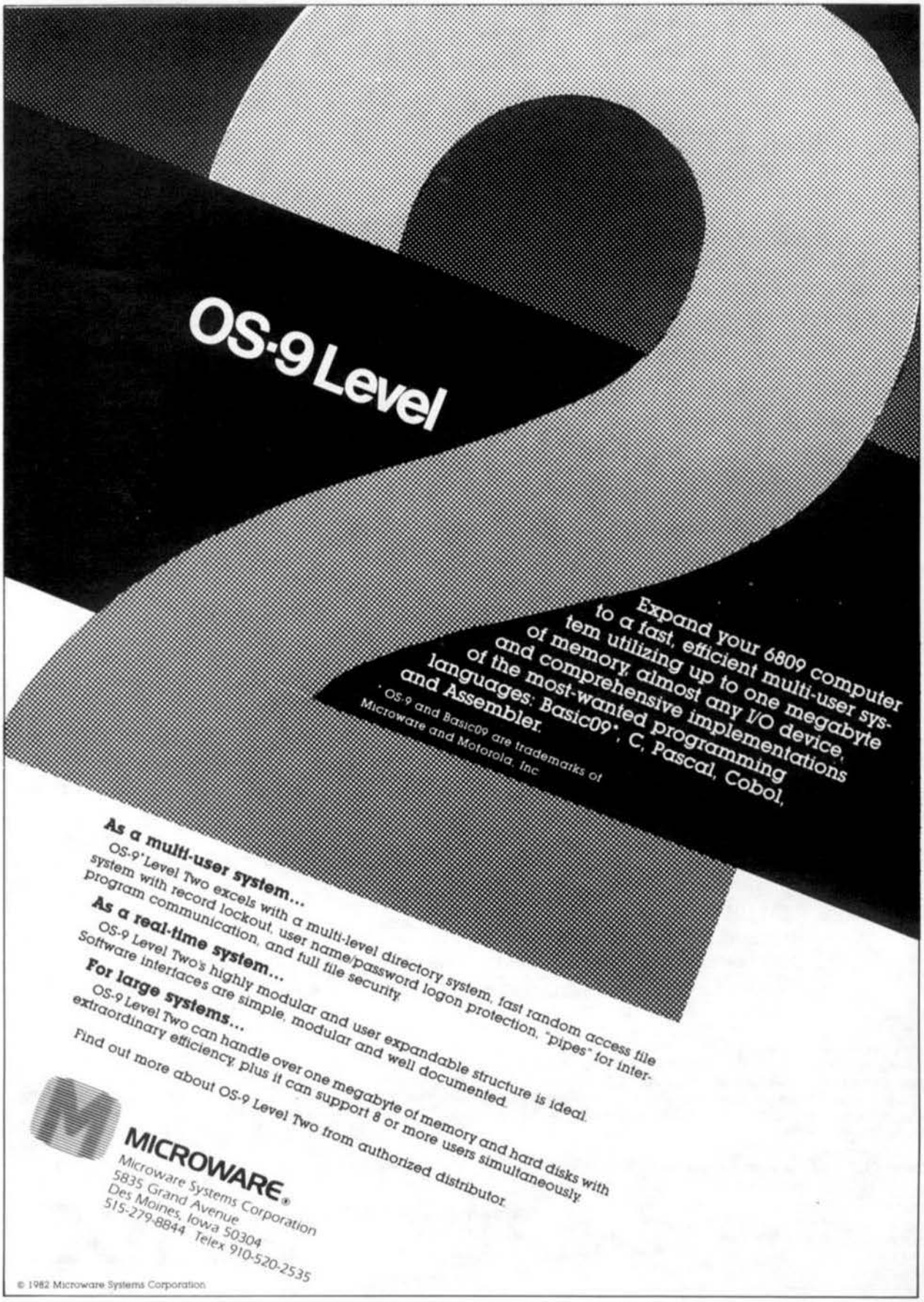
#90 includes one 19MB Drive, Interface, and Software **$4288.90**
#91 includes two 19MB Drives, Interface, and Software. **$6688.91**

Contact GIMIX for systems customized to your needs or for more information.

50 HZ Export Versions Available

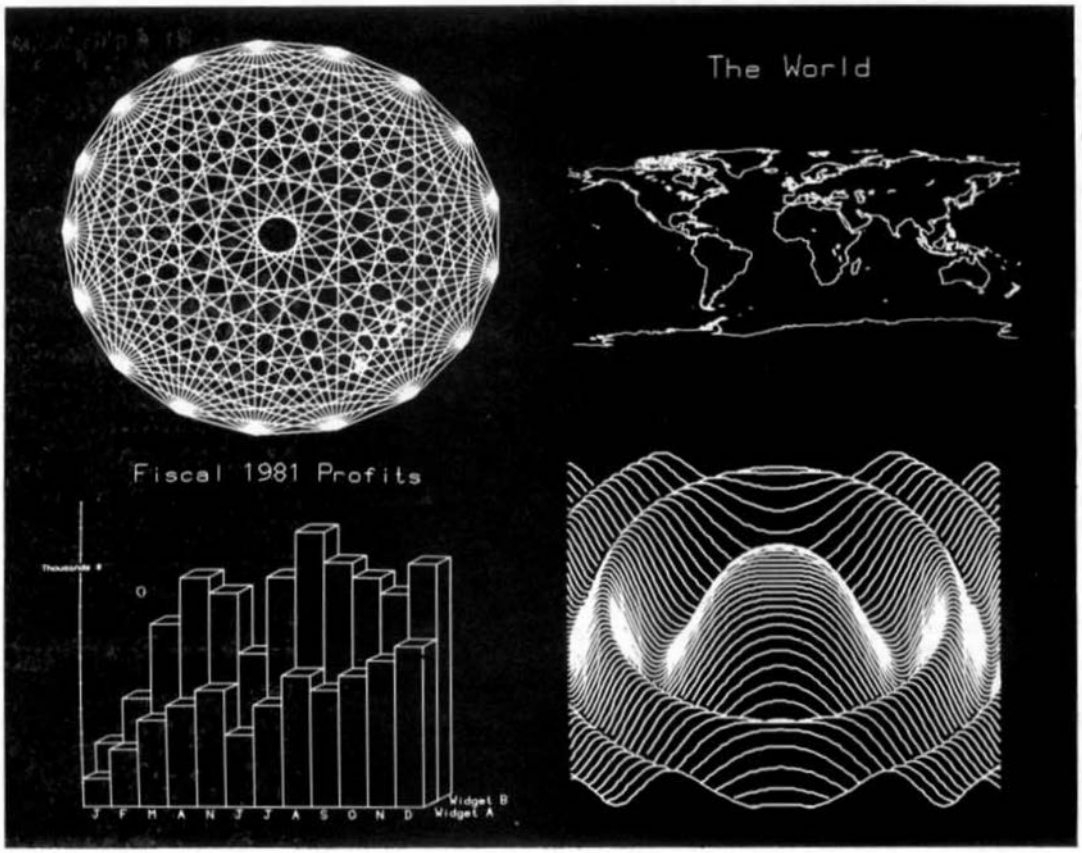
1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510
TWX 910-221-4055

GIMIX INC.


OS-9 Level
Expand your 6809 computer to a fast, efficient multi-user system utilizing up to one megabyte of memory, almost any I/O device, and comprehensive implementations of the most-wanted programming languages: Basic09*, C, Pascal, Cobol, and Assembler.
* OS-9 and Basic09 are trademarks of Microware and Motorola, Inc
As a multi-user system...
OS-9* Level Two excels with a multi-level directory system, fast random access file system with record lockout, user name/password logon protection, "pipes" for inter-program communication, and full file security.
As a real-time system...
OS-9 Level Two's highly modular and user expandable structure is ideal. Software interfaces are simple, modular and well documented.
For large systems...
OS-9 Level Two can handle over one megabyte of memory and hard disks with extraordinary efficiency, plus it can support 8 or more users simultaneously.
Find out more about OS-9 Level Two from authorized distributor.
MICROWARE®
Microware Systems Corporation
5835 Grand Avenue
Des Moines, Iowa 50304
515-279-8844 Telex 910-520-2535
© 1982 Microware Systems Corporation

ANNOUNCING ElectroScreen™ the Superior Alternative to the Traditional Alphanumeric Terminals

only $595



The ElectroScreen™ Intelligent Graphics Board Features:

Graphics

- 512 x 480 resolution bit-mapped display
- Interleaved memory access — fast, snow-free updates

Intelligence

- 6809 on-board mpu
- 6K on-board firmware
- STD syntax high level graphics command set
- Removes host graphics software burden
- Flexible text and graphics integration
- Multiple character sizes
- User programs can be run on-board

Terminal

- Terminal emulation on power-up
- 83 characters by 48 lines display
- Easy switching among user-defined character sets
- Fast hardware scrolling

Additional Features

- SS-50C and SS-64 compatible board
- Board communicates with host through parallel latches
- Composite and TTL level video output
- 8 channel 8 bit A/D converter
- Board occupies 4 address bytes

See your dealer today!

The ElectroScreen manual is available for $10. credited toward purchase of the board.

The ElectroScreen has a 90 day warranty from purchase date.

Dealers, please contact us for our special introductory package.

Privac Inc (703) 671-3900
3711 S. George Mason Dr., Falls Church, Va. 22041

Flex User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

SOME THOUGHTS

I just prepared the last column for mailing, and since Christmas is rapidly approaching, I thought I would get a good start so I don't have to spend the holiday week worrying about getting a column done.

Some thanks are in order. The December 1982 issue of '68' Micro Journal arrived today. I would like to thank Don Williams here for the very kind review of my book. I hope it lives up to Don's review in your mind if you get a copy. Incidentally, since it was my first book, it was done on a fixed price deal with TAB, so the number sold won't influence what I get out of it, except to make TAB receptive to my writing another book if sales go well. I value the comments of readers concerning the column and what I write in it, and I would also value any comments on the book "From BASIC to Pascal" if any of you would care to send critical comments. Do you think the presentation was out of balance, too deep, too shallow, confusing, covered everything but.... etc.

Forth Again

Just a couple of days ago I received a large amount of material from Ray Talbot showing some Forth applications in what I would call Real World systems. I've not had a chance to study all the material, but I will do so and come back with a report on what I find.

Along the same subject, I received a call from Manfred Peschke today. He tells me that his company has written a very large machine control program in Forth. He is sending me some further details. I will report on that also. It is becoming apparent that there are people out there doing significant software with Forth. They all insist that their programs are readable and that more than one person CAN work on a Forth program without great difficulty. Manfred said that the Forth words they write tend to be very short, no more than four lines or so, and that the "rules" are to document the parameters passed in and out on the stack, along with a line or two description of the word. He also indicated that because Forth words tend to be used so frequently, the end result object code is SMALLER than the equivalent program written in assembler. His estimate is that the code is about 80% as large as the assembler code to do the same thing. He said that there are considerable calculations involved in his program. I suspect most of them are integer calculations.

My estimate for Pascal is that the end code is two to three times as large as the equivalent Assembler program. Forth fans out there, don't all jump up and down too hard yet, even Ray Talbot admits that Forth isn't good for every kind of program. He particularly indicates that it might not be *quite* what is wanted for Scientific Calculations. I'm currently working with Forth in an attempt to write some standard words for my applications. If I am successful, I can see the possibility of trying Forth for one of my machine programs in the future. My programs in that general area tend to run about 10 pages of Pascal and a couple pages of assembler hardware drivers. It ought to be possible to do it with about three or four pages of Forth program.

I have received a letter from Gary Bergstrom concerning my January column remarks on Forth, and I thought you readers ought to hear a Forth user's side of the story too, so I am quoting it here, essentially intact, and I will add my response.

" Dear Ron,

Your comments on Forth are a little misleading ('68' Micro Journal 1/83), and as I said a year ago, yo carry more influence than you think. People see someone saying negative things about something and even if the writer changes his mind later and prints nice things the damage is done. (Imagine i had a column and said that Pascal was worthless etc. etc. and then later said - never mind. What would the reader remember? Think about it.)

Forth was not meant to be an end all language. As originally designed it was a controls language which was easily extendable (and except for LISP, it seems to be the only language you can extend). At this (controlling) it shines and you cannot cannot touch it for speed, conciseness, or legibility. But it is still a young language having spent most of its time to date in radio observatories and the like, and only recently emerging into the light of general programming. Many of the things that you say you must write in order to do what you want are already available in the public domain and more are becoming available daily.

Take for example floating point. You said that "there is no floating...". That is not the reason that traditionally Forth has not had floating point, and please note that most Forth's today have floating point as an option. The reason for the lack of floating point was speed. Very simple. Control programs typically are doing things in real time and hence optimized fixed point is used. I don't see how you could spend "considerable time learning it" and not realize this.

Also your example of a mailing list program uses a feature of Pascal that is typically not included in basic Forth packages - file handling. I feel that this challenge of <15 hours to develop in Forth is a little silly but I think that it could be done including the generating of a file handler adequate for the job (assuming that you wouldn't let me use one of the public domain Forth programs which handles files). But i don't have 10 hours to spare. All of this ignores the power of Forth. My Forth kernal doesn't have strings but I have a couple of screens that add strings; it doesn't have a case statement but I have a couple of screens that contain 4 different kinds of cases - a different kind for each type of application (ah, I hear it now - Why have four kinds when my Pascal can do everything with just one? Sure, and we can build computers with just NAND gates. If you don't want to learn four just learn one and ignore the others and you can still do as well as Pascal but if you want performance then don't refuse to continue to speak in a language without becoming fl ent in it). It doesn't have multidimensional arrays but again I have a screen that does; etc. etc. I can extend my Forth with software from myself and those publications which support it.

This doesn't begin to touch upon the true extensibility of Forth using the CREATE DOES> construction. This discussion can go on for a long time with someone who doesn't appreciate it and go nowhere.

Pascal is nice and its file handling c pability is equal to any but it cannot be changed. Procedures can be added but they are not the same as being able to add logical constructions. It is fixed. Forth can be used by the novice in the same way and would appear to have no advantages but let that person use it more and more and they will realize the power.

I use Basic when it is the best tool. I've used APL when it was. And Fortran, and LISP, and Assembler. In fact I just finished writing a 12K program here at work for the 6800 all in assembly language. I use Forth at home and we will be switching to Forth or "C" here at work to improve productivity ("C" runs faster, takes more ROM space, is slower to write, and less interactive than Forth, but this is a higher volume control application and usually speed comes first and development time and convenience is second).

As an aside - when I'm working on say a short (<5 screens) Forth program and discover a typo on a screen, I can run the editor, recompile and execute the program in less than 1 minute. To do the same thing on my FLEX system in "C" takes about 20 minutes (this would be only about 12-15 minutes if I had 2 disks but I only have 1). Now I know our programs are supposed to work the first time but you'd be surprised at how much time you can spend waiting for the machine to complete an assembly. I spend almost no time waiting for the Forth."

My Response

Gary, after my initial irritation with you for your quickness to send me a "nasty letter" as compared to your slowness to write an article for '68' about how Forth is so nice, I got down to business and wrote you a response. I decided that it would be fair to air your letter and my response in this column. I don't want to debate endlessly the merits of particular languages (or their lack of merit). I never ever called Forth "worthless" as you imply in the first paragraph of your letter. Also, feel free to write anything you like about any computer language. I give our readers credit for enough brains not to take any single opinion as absolute truth, particularly when it is clearly labeled as opinion. All of the languages serve a purpose. They have a slot in

the scheme of things and each is better at solving a particular type of problem.

I have probably given the impression in this column that I am a Pascal Nut, and that I feel that nothing else is any good. Nothing could be farther from the truth. I have said in previous columns that I use BASIC frequently to work out a particularly complex calculation, and then translate the BASIC program to Pascal. I do that just for the reason you cited in your letter. BASIC is interactive and I can change a line and try running it again.

I have written a program of about 6K in Assembler for a machine to balance crankshafts (6 different kinds) automatically, calculating from the unbalance readings on which counterweights and at which angles to drill standard diameter holes how deep, to balance the crank. It doesn't take much more than High School Trigonometry and a bit of Integral Calculus to solve the problem of "resolving" the unbalance vectors along the axis of the crank so that material can be removed at counterweights rather than in thin air. For this program I had to write my own Trig functions in Assembler.

Before the first usable Pascal came along, I had purchased and tried an early BASIC compiler (a five page program compiled into enough object code to overflow my limit at that time of 48K). I bought a couple other BASIC compilers and found them unacceptable for their rather arbitrary "extensions" to BASIC which is not very standard in the first place. I wrote the control logic for a machine (the replacement for a relay logic control system) in a version of PL/M that was then available for the 6800. It ran fast and worked fine, but the PL/M implementation had only integer variables so I couldn't use it for my complex calculations. The company bought the A/BASIC compiler, which also worked fine for integer applications (of which we had very few).

When Lucidata's implementation of Pascal came along (with REAL variables) I bought it and was able to learn enough Pascal in two days to be able to translate a BASIC program of about 300 lines to Pascal and get it running. A couple years ago when OmegaSoft released their Pascal, we bought it and found that it was faster than any other compiler we had ever tried. It is cumbersome to use. It generates 8 or 10 files for every program that you write. The compiler is reasonably fast. The assembler is very slow. The restrictiveness of Pascal is a PAIN. However, the programs are extremely readable (even after a couple years). The documentation is complete so that someone else can pick up a program and understand it. The point is that we use it because it is the best thing we have found to date to solve the sort of problems that we have. I have been involved for over a year in the writing of a Pascal program to control a multi axis machine tool. I have written approximately 1/3 of the entire program. My part was done entirely in Pascal. Another programmer wrote about 10K of drivers for the hardware interface in Assembler. A third programmer wrote a multi tasking system in Assembler, and the complete operator interface (a number of display pages with input by cursor positioning and numeric key pad) in Pascal. The whole thing occupies nearly 96K of EPROM.

I am not a staunch Pascal fan. I said once in a column that I didn't care what language I had to use to write my software, as long as it produced code that was reasonably compact and had acceptable execution time. When I find s thing better I will use it.

Let me quote from your letter above. "At this (controlling) it shines and you cannot ... touch it for speed, conciseness, or legibility." A few paragraphs later you say "C runs faster". I have a sort of benchmark program that I use to try all the compilers I get a chance to test (my prime number program). The Forth version doesn't run as fast as the one written in ABASIC, the PL/M version I mentioned above, or two of the four Pascal compilers I have tested (even after Talbot Microsystems wrote a special unsigned divide for the Forth version, that runs faster than the normal signed divide).

Getting back to the quote above, no one in his right mind would argue that Forth is not concise. So is shorthand, but not too many people can read it. I realize that Forth can be written legibly, but I'm told that avid Forth programmers look upon the use of named variables with displeasure, and would rather have several anonymous parameters floating around on the stack to save execution time.
That doesn't help readability at all.

You said that Forth is extendable as is no other language but LISP. I suspect you will say that I don't understand your definition of the word. I take extensibility to mean that you can write functions in the language that are "called" exactly like the functions already available in the language. By that definition, Pascal and "C" are as extensible as any language. In Pascal one gets the value of the sine of an angle by a statement such as VALUE := SIN (ANGLE); A function can be written, (for example, the Sine squared function) so that it can be used in exactly the same way. VALUE := SINSQR (ANGLE); By my definition that is extensibility. In "C" that function may be made a separate program module that may be pre compiled and simply linked to the program if it is needed. Some versions of Pascal allow the same sort of pre compiling and linking. I appreciate the fact that my extensions don't become an integral part of the language, and think that is a definite advantage!

You thought my example of a mailing list program was "silly" because I ought to know that file handling is not a standard part of Forth. THAT WAS MY POINT!

I've read all the arguments about why Forth doesn't have floating point (including the chapter in Leo Brodie's book "Starting Forth"). The example program given there is very nicely worked out. The difficulty is that the problems I have to solve don't fit that approach very well. I might have an operator input a dimension to four decimal places, ie. 1.3756 inches. I have to accommodate incremental encoders with pulse counts as high as 200,000 per inch. Numbers at both those extremes might be present in the same equation!

I am a bit surprised to learn that Forth has some floating point packages available. I've not seen one offered as part of a Forth for the 6809, though I have three Forths. Perhaps that would enable me to use Forth effectively to solve my programming problems.

Gary, you have one excellent point that I can't debate for a moment. The compile times for the versions of Pascal and C that I have are long. Unfortunately the best compiler in terms of programming to solve the problems that my programs must solve, is the slowest. Forth compiles like lightning!

I think the crux of the problem is that you (Forth users collectively) think the best feature of Forth is that you can make it anything you want. To me that is a disadvantage. I think the compiler should be a STANDARD thing, and that the user program should be the only variable. In Forth, the compiler and the application tend to merge into one entity. In fact, when you write an application, you are essentially just extending the compiler.

As I have said every time I write about languages in general and Forth in particular, If you like Forth, USE IT and LOVE IT. Just don't tell me that everyone who doesn't like it is stupid.

Norm Commo has written a column for this magazine for some time, dealing with the "C" language and the presently available implementations. Why doesn't s one of you avid Forth users do the same with Forth? Sorry, but '68' doesn't pay anyone who writes articles for them. Still interested?

MORE ON SPELLB

I just received a "final" version of SPELLB, which I reviewed several months ago. I decided to run a few more checks, so I ran the file that I have been using all along, one with approximately 3900 words in it. Time from command to list of suspect words was 3 minutes 26 seconds compared to 3 minutes 44 seconds reported with an early version. The suspect list was 16 words long, and NO valid words were contained in it.

Since I had reported all the valid words I decided to try a couple other files. The largest one in the book is 118 sectors or very nearly 5000 words. It took 4 minutes 12 seconds to produce a word list that contained five words that I consider valid. They were "quadrants", "radians", "redone", "incremented" and "decremented". By use of the HELP facility, I found that the dictionary does contain "quadrant", "radian", "increment", "incremental", and "decrement". If you want to argue that "redone" is a coined word, I won't be upset. If I were not certain that I had spelled "quadrants" correctly, the use of HELP would show me the word "quadrant" and I would n d no further help, so I could eliminate that without resorting to a hard copy dictionary. Same goes for "radians", "incremented" and "decremented". When the same word in a slightly different form appears in the dictionary, there is no need to go look it up!

I tried two other files of about 2800 words each, and SPELLB reported NO valid words in one of them, and three in the second. They were "unterminated", "sparingly", and "referring", as in "The transmission line was unterminated. If you use SPELLB you will find yourself referring to the dictionary very sparingly". The SPELLB dictionary contained "reference" and "spare", but nothing like "unterminated". I wouldn't be upset if someone told me that "unterminated" is a technical word that belongs in my "special dictionary".

Dan Farnsworth of Palm Beach Software assured me that he would try to make the dictionary comprehensive enough so that no valid words would remain in a suspect list. I think 8 words in a total of nearly 15000 comes admirably close to that goal. By my calculations, my 1 MHZ system with 8 inch disk drives allows SPELLB to clip along checking just about 1150 words per minute! That number may vary somewhat for another writer who uses a different mix of common words and unusual ones, but for my texts it was rather constant, varying only a few percent from one file to another. I rated this software highly in my review, and I can honestly say that the continuing efforts to improve it have been successful.

Heating Control

Last time, we had started the discussion of a home heating control to illustrate the use of a 6809 system to perform a control function. We had gotten as far as a program to use the JPC A/D board to read temperatures and temperature commands on up to 8 rooms (16 channels) and display the readings and commands on the terminal. We also showed the schematic for the hardware required for that portion of the control. In order to get this out on time or nearly so, and due to the length of the Forth discussion above, I am going to defer that subject until next month.

COLOR User Notes

Robert L. Nay
5900 Cassandra Smith Rd.
Hixson, Tn. 37343

WHAT WE HAVE HAD THE MOST REQUEST FOR FROM THOSE WHO DABBLE WITH THE 'REAL THING'

I'll keep this short! The listing provided this month is a "fairly" (I would guess 99+%) Cross Reference Listing of the Color Computer V1.1 BASIC ROM. The arrow pointing IN in the first column indicates that that location is a Subroutine Entry Point. Note especially location $B2CE; it is an Indirect Subroutine Jump off of the X Reg. Also note that there are no calls from within the ROM to the initial Input Vectors beginning with $A000 (and no need for them). Finally, ALL of the Hardware Vectors are lumped together at LBFFA; I'll try to sort them out for you next month.

If you come up with any additions, let me know.

```
BASIC V1.1 ROM Cross Ref Listing

L0000   AD8A AE11 AE27
L0001   AEEC AEF3 AEF5 AF6F AF7F B073 B07C B087
        B2DB B2E7 B2ED B51A B52B BD04
L0002   B08B B2E9 B2EF B51C B52F
L0003   AC80 ACD1 B386 B3E3 B41D B441 B490 B495
        B4A2 B4BB B8A0 B8C4
L0004   AF10 AF26 AF33
L0005   B35A B401 B426 B43D B464 B49C
L0006   AF93 B06D B148 B18D B1B8 B1C2 B226 B289
        B309 B367 B37F B403 B424 B4F4 B562 B6B8
        BF2C
L0007   B36D B587
L0008   AF5E AD43 AD49 B387 B38F
L0009   AFDF B04A B05D B0D7
L000A   B20E B342 BFA4
L000B   AD3A B54C B56A B5A0 B679
L000D   B564 B675 B67D
L000F   A8FD AC11 AC1A AD53 B479 B478 B490
L0011   ACD7 AC00 AC16
L0012   BA97
L0013   BA02 B805 B81C B821 B898 B8A8
L0014   BAD4 B816 B81A B823
L0015   BAD6 B810 B814 B825 BC0F BF6D
L0016   BAD8 B80A B80E B827
L0017   AC3C AC3F B4D4 B4E2
L0019   A082 A474 A4D6 ACEF AD03 AD19 AD21 ADE4
        AED4
L001B   A47E AAEB ACBA ACBC ACC4 ACCD ACE7 AD1F
        AD2D AE67 AFAD B391 B5A8
L001D   AD2F B395 B3BF B3CC B428 B5AA
L001F   AC33 AD31 B3B6 B3C8 B42A B489 B496 B4F0
        B5B6
L0021   A09D AD3A AE6B AFA6 B576 B599
L0023   A097 AD28 B572 B578 B57B B593 B5DE B5F8
        B663 B669 B668
L0025   B581 B643 B631
L0027   A095 AD26 AE4B AE6D B591
L0029   AE1C AE3C
L002B   ACA8 AD01 AEAD AF69 AF71 AF78 AF83 B053
        B08C B736 B739 B7B7 B7A3
L0038   AD3F AE20 AE34
L002E   AD41
L0029   ADA7 AE0F AE1E AFF0
L0031   AFE3 B0CB
L0033   AD6B B04a
L0035   B04C B057 B0AA B0D5
L0037   B38C B393 B3D0 B408 B410 B42E B434
L0038   B3BD
L0039   B3DC B4CB
L003B   AC09 AC1B AD96 AEC4 AF8C AFC9 B051 B1D1
        B117
L0035   B1AB B201
L003F   B146 B173 B177 B178 B1B1 B1C0 B1DA B308
L0040   BC2F BD49 BEF9 BF2D
L0041   AC20 AC05 ACE5 B3BD B3CA B45F B494 B498
        B382 B384 B3BC B3BE B3CA B3FD B896 BAB8
        B807
L0042   B846 B87C B882 B893 B8B4 B8EA
L0043   AC24 ACCF B390 B3F9 B829 B831 B842 B8E4
L0044   B828 B84E B8CC

L0045   AC30 ACE1 B3CA B4D0 B466 B604 B608 BC2A
        B01E B067 B088 B088 B0FF BE14 BE1B BE27
        BE3A BE7E BF17
L0046   B961 B988
L0047   AC2C ACBA AD14 B3C1 B3CE B598 B5E3 B5EA
        B5F7 BD1C BD5D BD65 BD69 BD72 BD7B BDA5
        BDA9 BDB2 BE38 BE9A
L0048   BD53 BD59
L0049   B597 B5E0 B5EF B602
L004D   AF84 AF8E AFC3 B500 B633 B6BA B6FC
L0047   AF22 B1EA B220 B3ED B744 B9D2 B9D8 B9DE
        BA3A BA51 BA57 BA66 BB45 BB4B BB53 BB78
        BB93 BB97 BC26 BC35 BC50 BC5F BC6D BC86
        BCA0 BCC8 BCD7 BCEE BD00 BD16 BDEC BEE9
        BF68 BF8C
L0050   A074 A076 D1EC B4F6 B609 B813 B818 BA23
        BA4D BA6A BA78 BA8B BABF BAF2 B8A4 B8F2
        BC08 BC1C BC30 BC7C BC82 BCAA BCE9 BD09
        B8CC BE62 BE64 BF3E BF63
```

```
L0051   BA09 BA00 BA21 BA27 BA4B BA6C BA78 BA8A
        B8BA B8BC B8C1 BC58 BCa3 BC7E BC80 BD00
        BD18 BE3C BE60
L0052   AF44 B1EE B287 B2B0 B3F8 B412 B46A B497
        B5aa B66F Ba1F Ba37 B74D B803 B807 BA2B
        BA2D BA49 BA6E BA7F B883 BA83 BAE2 B8B0
        B8Ea BC11 BC24 BC45 BC8C BC8A BC8e B8D0
        B81A BE3a BE3A BF43 BF5A
L0053   A5aD A997 AF58 B694 B6D1 B6ED B6F1 B9FD
        BA01 BA29 BA2F BA47 BA7o BA81 BADA B8Ba
        B8E0 BCa7 BC8C B002 B00F BE30 BC54
L0054   AD7D B115 b1E7 B21C B328 B330 B3E9 B740
        B98C B90C BA3C BA78 BB3Y B85Y B83C BC1B
        BC39 BC4C BC5W BC71 BC8E BC93 BC9C BCE4
        BCDE BCEa BCFB BCFA BD14 B00E B08B BEED
        BF67 BF97 BFA0
L0055   B829 B87F BF08 BF1A
L0056   A573 AF8D B310 B319 B328 B513 B53B B558
L0058   B312 B32A B811 B52D
L0058   BAAA BC02 BC81 BC84
L005C   B212 B301 B9CA B843 BC4E BCa1 BF8A
L0050   B214 B2F0 B2FF BA11 BA1E B835 B8A6 B8D6
        B8F0 BBF6 B948
L0050   BA08 B81B B9AC B8D4 B8EA B8E2 BC54 BCa5
L005F   B21a B314 BA02 B912 B83F B907 B802 B8E4
        B8EB BC3A BF4D
L0050   B9FF B80C B880 B8D0 B8DE B8E2 BC69
L0061   B21A B2F9 B9DA B831 B837 BC4A BF7E
L0062   B21E B520 B53D B547 Ba1B Ba2D B639 B9BE
        B9EC B93B B857 B873 B88W BF88
L0063   B9FB Ba7D BA31 BA45 BA9E BAa2 B89C BABO
        BA0E BAEa BAEE BAF6 B807 B8FY BC1E BC97
        BCBC BCF6 BF4F
L0064   BC49 B452 B471 B4Aa B481 B40E B539 B8E6
        B83E BEF0 B8FF BF04 BF0A BF0E BF17
L0065   B4DC
L0066   B76C B788 B79F B7AE B7B2
L0068   AABa AAA0 AaF0 ACa0 4C82 AB61 4D8C A615
        AE3E AE99 AEAF AEDC AF65 AFF7 B1 A 90CA
L006A   A37C B977
L006B   B9A0
L006C   A37E B961 B968 B979 B98A
L006D   A3B0
L006E   A364 A3B4 B081 B93F B9BB
L006F   A069 A17B A2B7 A3a6 A3A1 A3F0 A409 A42B
        A430 A432 A463 A4B5 A54C A5B7 A901 A5D0
        A5E6 A608 A60A A6AF AC55 AC96 AE22 AF08
        AFFF B023 B760 B79A B8FB
L0070   A179 A183 A39D ACB6 B041
L0071   A017 A0E4 A100
L0072   A01D A0ED
L0074   A093 AE36
L0078   A292 3F7 60E A439 A449 A46F A498 A4B7
        A633 A668 AaSF A670
L0079   A17F A190 A297 A2A4 A2B1 A43D A5BF Aa44
        Aa5a
L007A   A188 A18E A29E A2A2 Aa55
L007C   A2AA A471 A491 A4E5 A535 A63A A672 A698
        A6E7 A71B A721 A805 AB11
L007D   A2B3 A67C A688 A693 A6A0 A67a A71F A72B
        A7F6 A7FA A815
L007E   A2AF A678 A6B0 A6D0 A6DA A529 A533 Aaaa
        Aa88 A70F A7FE A809
L0080   A723 A733 A735 A73D A807 A824
L0081   A6ED A708 A727 A737 A74a A7FB A820
L0082   A74B A750 A780 A78C A78E A79F A7A1 A7B8
L0083   A737 A75B A76C A7A7 A7AB A781
L0084   A75F A794
L0085   A82E A855
L0086   ABA8 A88A A88C A8F2 A8FD
L0087   A393 A5a4 A5a8 ADF8
L0088   A1A1 A1BA A30C A321 A344 A561 A6BE Aa94
        A92B
L0089   A173
L008A   A46A A505 A7D1 AF67 B0FA B1F4 B3A8 BC88
        B012
L008C   A948 B98A
L0050   A934 A970 A988 A9C2
L008F   A0A4 A75A
L0090   A7B3
L0091   A7B7
L0092   A7DC
L0094   A199 A19F
L0095   A302
L0099   A36D
L009B   A2E3 A36F
L009C   A2DF A2E1 A2E7
L009D   A51D A543 A545
> L009F   A4FE A5A9 A78F A992 AA22 ACBE AD8B ADC0
        AD08 AE80 AF2D AF37 AF5D AE85 B04B B13D
        B17D B228 B275 B290 B295 B3A9 B373 B383
        B3E4 B709 B77A B997 BD31 BD3D B055
> L00A5   A41D A44F A507 A586 A5C7 AD85 AE05 AE4A
        AE7B AEA4 AF17 AF39 B09E B0B0 B137 B168
        B352 B358 B35E B417 B6D3 B714 B728 B762
        B772 B911 B945 B94C B954
L00A8   ACBC ADA5 ADBE AE0D AE3A AE97 AE8D AEDE
        AEE5 AEEF AFF2 B053 B069 B071 B0AB B0AE
        B089 B12F B1C6 B244 B249 B26F B71D B71F
        B731 B824 B8A8
L00AB   BAF8 BF5C
L00AC   BAF0
L00AB   BAE8 BF32
L00AB   BAE0
L00BC   B958
L0100   A3B6 BFF2
L0103   BFF2
L0106   BFFA
L0109   BFFA
L010C   A0AC BFF2
L010F   BFF2
L0116   B7E6 B8W8 BF38 BF62
L0118   BF40 BF37
L011A   A304 A328
L011B   A200 B8D4
L0123   ADD1
L0128   B2CB
L012D   ADED
L0132   B2W8
L0152   A1CB
L015A   A9D4
> L015E   A08A A3F6 A9E0
> L0161   ABB9
```

```
> L0164  A33F
> L0167  A282
> L016A  A176
> L016D  A3E0
> L0170  A406
> L0173  A42e
> L0176  A42D
> L0179  0 10
> L017C  B061
> L017F  A549
> L0182  A390
> L0185  A4BF
> L0188  A5CE
> L018B  B223
> L018E  AC46
> L0191  AC49
> L0194  AE75
> L0197  B022
> L019A  A09E
> L019D  A8C4
> L01A0  A410
> L01A3  0021
> L01A6  B7C2
  L01A9  AD33 B59D
  L01D1  A578 A590 B54E
  L01D2  A66C A69F
  L01DA  A24C A581 A632 AAA3 A681 A69C
  L01E2  A4CB A511
  L01E3  A4AB A62B
  L01E4  A44a A51F A62E A6D1
  L01E5  A51B
  L01E7  A524
  L02DB  ACDA
  L02D9  A0C3
  L02DA  ACAA
  L02DB  B0A5
  L02DC  A3D4 ACD0 B01E B02A
  L02DD  A3e5 B7B6 B7C7
  L03D6  B014
  L03D7  A00E
  L03D8  B508
  L03D9  B500 B53F
  L03DA  BDD9 BE8C
  L0400  A212 A36B A6F3 A6FD A800 A92A
  L0401  A074
  L049E  A1AB
  L05C0  A3E3
  L05FF  A34a A 31
  L1010  A377
  L1A01  B145

  L2500  BA58
  L2B9a  B14E
  L3404  B159
  L4558  A0CB
  L8000  A0CE
  L8002  A0D1
  L8602  A743
  L8608  A975
  L8621  B7DF
  L863F  B9AE
  L8804  AEE2
  L8899  B2C6
  L8D9B  BB8E
  L9E3B  8C32
  L9E97  A304
  LA000
  LA002
  LA004
  LA006
  LA008
  LA00A
  LA00C
  LA00E  6027
  LA027  BFFA A027
  LA02A  A105
  LA05e  6059
  LA070  A0AC
  LA072  A0a6
  LA074  A01B A023
  LA077  A07B
  LA084  A041
  LA093  A0EB
  LA0C0  A6C3
  LA0E8  A0DD

  LA0F3  A0Ea
  LA3FC  A0F9
  1A10B  A102
  LA10B  A0A1
  LA146  A0D7
  LA1A5  A939
> LA171  A39A
> LA176  A171
  LA18a  A181
  LA197  A192
> LA1   A1B3
  LA1A9  A1FB
  LA1B1  A170
  LA1B3  A1B7
  LA1BF  A1D8
> LA1C1  A000 A1B5 A5AB A0E9 A0F8
> LA1CB  A1C3
  LA1D5  A1E9
  LA1F1  A1F6
  LA20B  A202 A207 A29D A2A1 A2A8
  LA21A  A24D
  LA220  A218
  LA229  A1D4
  LA22B  A222
  LA22C  A22a
> LA22E  A200 A254 A2A4
> LA237  A215 A21C
  LA238  A246
  LA243  A23F

  LA246  A1FC
  LA25C  A258
  LA263  A24B A252
  LA269  A266
> LA282  A002 A3B  A3E8 A6B1 B7B1
  LA29E  A29A
  LA2A6  A29B
> LA2AB  A29C A441
> LA2B6  A446
  LA2BF  A2BC
  LA2CA  A2D5

  LA2E7  A2D0
  LA2ED  A2E3 A2F1
> LA2F8  A2C3 A2D7
> LA2FD  A2C6 A2D0
> LA302  A300
> LA3D5  A2E0 A2E8
  LA30A  A2BE
  LA31D  A310
  LA323  A328
  LA32F  A31F
  LA340  A338
  LA342  A334 A33C
  LA344  A31B A32D
  LA34E  A356
  LA350  A315 A331 A349
> LA35F  B07E D 3C B95C B966 B997
  LA373  A3a8
  LA37C  B371 A3B9
  LA3B4  A3aB
  LA3BD  A3A7
> LA3D0  A3AE A3B9 AC7C B035
  LA39a  A3DB A3DF A3E3 A3E8
  LA3B4  A3AB
  LA3BB  A3C0
  LA3C2  A3B6
  LA3C8  3A3
  LA3CC  A39F
  LA3CD  A3C6
  LA3D9  A3CA
  LA3EB  A3B2
> LA3ED  0587 B009
  LA3F8  A410
  LA400  A3F9
  LA403  A3F3 A413
> LA406  B90E
  LA415  A5F2 A401 A40C
  LA41B  A424
> LA42a  A416 AE02 AE7B
> LA470  A418 A4C2 B797
  LA444  A43F
  LA449  A43B
  LA44B  A41F A435 A45B
  LA469  A451

  LA47B  A48F
  LA48C  A4 B
  LA491  A482
  LA48F  AC8B
  LA4C8  A4A9
  LA4CD  A4AE A516 A522
  LA4D0  A8C9
  LA4D8  A8E9
  LA4F8  A6E3 A467
  LA4FB  A531 A637
  LA4FE  A49C
  LA511  A509
  LA52E  A539
> LA838  A4ED
  LA845  A53E
> LA589  B790
> LA554  B902
  LA56B  A56a
> LA57B  A64C A500 A9C9
  LA57F  A584
> LA598  AAAF
> LA59A  A049 A0B1 A59F
  LA5A1  A5AF A5BB A5e3
> LA5A2  A421 Ra01
> LA5AB  A418 B004 B908
  LA5AB  A5A7
> LA5AE  A3D5
  LA5C4  A5BC A3C9
> LA5C5  A4A0 A5EC A604
> LA3C9  A7C7 A99A
  LA9E4  A5D0 A5E1
> LA3EB  A90A
  LA616  A603 A4CB A625 A651
  LA619  A4FB A5F3 A638 A63C A64E
  LA61C  A64A A661
  LA61F  A5B4 A5BE A5C2
  LA624  A610
> LA635  A198 A4B9 A642
> LA648  A4A3 A502 A5EE A629
  LA650  A2BC A67F
  LA652  A6dA
  LA657  A627 A63E A639
  LA658  A614
> LA5EC  A45E
> LA6AF  A46C
> LA6B1  A64C
  LA6B6  A6C9
  LA69a  A6B9
  LA6B4  A6A0
  LA6C0  A6BF A6C5
  LA6D0  A69A A6C7
> LA6D1  A5F0 A6C6
  LA6D9  A600
  LA6DF  A6D1 A6E3
> LA6E3  A6D8 A6E1
  LA6E0  A6E9
> LA6F3  A700
> LA6F0  A6CD
  LA700  A6EA A6F9
> LA701  A635 A696 A6DF
  LA709  A6F1
> LA70B  A006 A6E0 A52E A609 A703
  LA712  A717
  LA72B  A739
  LA73B  A729
  LA744  A731
  LA74a  A73F
> LA749  A719 A71D A72B A73B
  LA74D  A792
> LA755  A712 A748
> LA75D  A755
> LA763  A765 A782
> LA767  A749 A7A9
> LA76C  A763 A767 A773 A777
  LA773  A761 A775
> LA777  A779 A7AF
> LA77C  A004 A4D3 A528 A6DA A701
  LA782  A792
  LA784  A799
  LA78B  A79D
  LA792  A7A5
  LA797  A7B6

  LA798  A78A
> LA7A7  A78B A797
> LA7AD  A784 A79B
  LA7B1  A7AB
  A7BA   A7B0
> LA7CA  A77E A7DA
> LA7D1  A0FC A0FF
> LA7D3  A1AE A210 A307 A705
> LA7D8  A00C A4 3 A7E3
  LA7DE  A7E2
> LA7E5  A207 A678
  LA7E7  A495
> A7 9   A53B A7 5 A7C3 AC4C
> LA7F0  A7CF
> LA7F4  A008 A48C A7E7
  LA800  A803
  LA805  A7FC
  LA81C  A8C2
  LA82A  A81A
> LA82B  A7DE A8D8
> LA72A  A80F A813 A817 A81E A82A
  LA82B  A858
  LA833  6044
  LA848  A839 A853
  LA855  A841 A84E
  LA85C  A833
  LA880  A833 A84A
  LA893  A8BE
  LA89C  A897
  LA890  A893
  LA8A4  A8A3

  LA8AC  A8B0
> LA8C1  A8B0 A8B1
> LA8C4  A8F3
  LA8D5  A8BB A8CC
  LA8EE  A8F0
  LA8FC  A898
  LA909  A8FF
  LA90A  A8F8
> LA908  A8DF A8B3
  LA923  A910
> LA928  A07D A0F0 A3B8 BABB A913 A637
> LA92D  A35A
  LA92F  A934
  LA937  A91A
> LA93F  A96F
> LA942  A94B
  LA948  A8D5
  LA964  A972
> LA974  A996 A9DE AC4F
> LA97a  A962 A9A0
  LA9B4  A44a
> LA9C5  A9a  A9aA
> LA9D7  A968 A9AE
  LA9BC  A98D
> LA9A2  A960  99E A9E9
> LA9A7  A9A5
  LA9B0  A9AC
  LA9C5  A9B6 A9BE
  LA9D4  A9D0
> LA9DE  AD0A A902
  LA9E5  AA17
  LA9EB  AA10
  LA9EE  AA0a
  LA9FF  A9FA
  LAA12  AA0C
  LAA24  AA20
  LAA2B  AA1C
  LAA31  B19A
  LABAF  AC5D
  LABE0  AC63
  LABE7  B0C9
  LABEC  A46F
  LABE9  AC7a
  LABF1  AE24
> LABF9  AD4E AEC4 B103
  L BFD  AC14
  LAC1a  AC08
  LAC1A  BC03 BC09
> LAC1E  AC07 B3C3
> LAC20  B9FF
  LAC2B  AC2E
  LAC32  AC4J
> LAC3B  A05B AB94 B160 B45C
> LAC37  A4DD AC1E B483
  LAC44  AC3A AE72 B47D
  LAC4a  A3F0 A621 AC90 AE36 AED4 AF0C B153 B279
         B44C B555 B627 BA94 BC08

  LACA8  AE2D
  LAC73  A0F3 A4C3 AC4E AE29 B79C
  LAC7E  A4BC AC84 AC90 ACED
> LACA0  AC61 AC63
  LACA5  AC92
  LACCD  ACC6
  LACC8  ACB4
  LACDD  ACE3
  LACE9  A4F3 ACCB
> LACE7  ACEB
  ACF1   ACF5
  ACF7   AD79
> LAD01  AC93 B769
> LAD05  AD10 AEB6
  LAD12  AD07
  LAD14  A90C A017
  LAD1a  ACF3
> LAD19  A9CB A480 A400 A4F8
> LAD21  ACE9 AE7D
> LAD26  AE6F AEB1
> LAD33  A0EB AC52
  LAD59  AD53
  LAD7F  AD7B
  LADV0  AD89
  LAD9E  A0C4 AEA1 B131
  LADB1  ADDE
  LApB6  A0AB
  LADC0  AC9D ADAF
> LADC6  ADC2 AF3F
  LADDC  ADCF
> LADE4  AD2A
  LADE8  B0D9
> LA0EB  A551 ADA3
  LADF0  AE00
  LADFA  ADEE
  LADFB  A0F6 A0FE
  LAE09  ADF2
  LAE0B  AE07
  LAE11  B03C

  LAE15  A0BE
  LAE22  AE1A
  LAE40  A0C6 AE08 AE30
  LAE5A  AE4C
  LAE6F  AE41
  LAE72  AE58 AE3E AE45 AEA9
  LAE8B  AF5A
  LAE9F  AE84
> LAE44  AE7C AE9F AF39
  AE86   AEB2
> LAE99  AD23 B1C9
  LAEBF  AEC0
  LAED2  AE89
  LAED7  AE9D AF52
  LAEDA  AEC0
> LAEE0  AF20
  LAEE7  AEF9 AE9F AF29
> LAEE8  A08E AEE0 B080
> LAEEB  AEA9
  LAEF1  AF05
  LAEF7  AF0E AF17
  LAF0C  AF08
  LAF22  AF1B
  LAF28  AF31 AF35
  LAF39  AF24
  LAF52  AF79
  LAF54  AF4E AF63
  LAF5D  AF3a
> LAF67  ACA9 AEAa AF3F B76a B77E
  LAF6B  AF07
> LAF9A  A06B ADC9
> LAF4a  B093

  LAFBE  AF4B AF4F
  LAFCE  AF6B AFEa
  LAFD6  B004
  LAFDC  AFFB B043
  LAFDF  AFDA
  LAFEA  AFE1
> LB002  AFF0
  LB00F  B004
  LB01E  B011
> LB02F  B027 B0A7
  LB03F  B03B
  LB049  B025 B02D
  LB04E  B087
  LB069  B058 B003
  LB088  B077 B083
  LB09B  B06F
  LB09E  B096
  LB0AB  B0A9
  LB0B9  B73F B0D1
  LBCCB  B0E1
  LB0D5  B0B2
  LB0E7  B0BF B0E1 B14C
> LB0FC  B0FB B13F
  LB101  B0FC
  LB10A  B0C7
  LB10C  B106
  LB131  B13B
  LB134  B126
> LB141  A5AB A0a0 A0BD AF14 B3EA B70B B73D
> LB143  AD6A B1A5 B1FE B2D1 B4FD
> LB146  B2B6 B616 B654
> LB14B  AF9D B2F4
  LB14F  B14A

  LB153  B10A
> LB156  A500 A5F9 AF97 B141 B264 B0B1 B929
> LB15A  B1DC B254 B27E
  LB168  B640
  LB16A  B17F
  LB181  B16C B170
  LB19F  B1C4
  LB1A7  B1D2
  LB1BB  B1B3
> LB1C6  B156 B1 B
  LB1CB  B1B0
  LB1CE  B1AF
> LB1D4  B1A9
  LB1DF  B179
> LB1E2  B1B0
> LB1E6  A093
  LB1EA  A07C
  LB1F4  B185 B1B0
  LB1FA  B143
  LB201  B1FC
  LB203  B1B6 B1D0
  LB220  B192 B1F8
  LB222  B207 B208 B2B9
> LB223  B163 B613
  LB228  B23E
  LB22C  B236
  LB22F  B226
> LB244  B013
> LB249  B090
  LB24E  B242
  LB25F  B030
```

```
> LB 67   A90D B41F B6F5
> LB26A   ABC1 B2A2 B2A9
> LB 6D   A453 A308 A3A2 A803 A93F AE4E B00C B6B4
          B2B4 B34B B6D9 B738 B915
> LB26F   A458 AD67 AF1F AF47 AF90 B018
  LB277   A5C6 ADB1 AED7 AFE7 B10F B273 B34  B9B3
  LB27C   B23A
  LB284   B232
  LB287   B3B0
  LB290   B260
  LB29F   B299
  LB2C7   B2A3
  LB2C9   B2A7 B2AF
>> B2CE   ==> JSR [0,X]
  LB2ED   B2E5
  LB2F1   B2EB
  LB309   B2F7
  LB328   B31F B323
  LB32D   B338
  LB334   B32E
  LB33F   B307 B332 B33C
  LB348   B344
  LB34B   B354
> LB357   AF89 904E B0FE B2B4
> LB35A   B350
  LB371   B36B
  LB373   B375 B379
  LB37B   B36F
  LB385   B37D
  LB395   B3A0
> LB3A2   B22F B360 B36B B377 B835 B8E3
  LB3AA   B3A4
  LB3AB   B397
  LB3DC   B39C
  LB3DE   B3B4
  LB3DF   B3F3
> LB3E4   A554 B40C
> LB3E9   AE43
> LB3E9   B70E
> LB3ED   A5AE B257 B208 B2E0
  LB3FB   B3F1
  LB401   B388
  LB408   B41B
  LB42A   B439
  LB43B   B433
  LB447   B4EB
  LB44A   A05A A948 A9C8 B3E8 B3F9 B706
  LB44C   B63F
  LB44F   B42C
  LB461   B477
  LB46D   B466
  LB48C   B492
  LB4A0   B445
  LB4A6   B4B0
  LB4B9   B4B3
  LB4CD   B49E
> LB4CE   B46F B485
  LB4D8   B4EB
  LB4E6   B4E0
  LB4EB   B4AF B4DA B4E4
  LB4F3   A9D8 B683 B734
  LB4F4   A3E9 B23C B2F1
> LB50D   AFB3 B949 B62
> LB50F   96B7
> LB511   B696
> LB516   B935
> LB518   B246 B308 B99C
> LB51E   B08D
  LB526   B531
  LB533   B52D
  LB537   B529
  LB539   B535
  LB543   B4F2
  LB54C   B243 B63D B69D
  LB555   B989
  LB558   B521
  LB56D   B5AF B691
  LB58F   B58F
  LB595   B579
  LB591   B588
  LB593   B60C
  LB5A0   B5A0
  LB5AB   B5A2
  LB5AA   B580
  LB5B2   B5AC
  LB5B4   B5C2
  LB5B6   B5CC
  LB5CA   B5B0
  LB5D2   B5AE
  LB5D8   B5A4 B5CE
  LB5EC   B506 B50A B5E1 B5E6
  LB5EE   B5F1
  LB5EF   B5B8
  LB60F   B18F
  LB62A   B623
  LB645   AF98 B631
  LB645   B949 B637 B6C3
  L 64A   B64F
  LB64E   B648
  LB654   A980 8486
  LB657   B390 B99F
  LB659   B316 B635 B63B B68C
  LB66F   B665
  LB672   B650
  LB675   AFC0 B65B
  LB680   B677
  LB683    6A2
> LB686   B681 B6A4 B716
  LB68F   B9AF
  LB69B   A575
  LB69D   B6C6
> LB6A4   A9FC 8AA0
  LB6AE   B6CB
  LB6B5   B6B0 B6E6 B6EF B6F3
  LB6DE   B6D7
> LB6F5   B6AB B6CB B6DE
  LB706   B8A6 B6E0 B712 B742 B748
> LB709   B97E
> LB70B   A8C7 A915 A942 AF42 B2BF B60C B73B
> LB70E   A9CA B68C
> LB734   B757
> LB738   A884 A8D0
> LB73D   A60E A540 AE31 B734
> LB740   B750
  LB764   B486
  LB7B9   B778
  LB7B4   B770 B77C
  LB7B9   B774 B7B1
  LB7BB   B79B
  LB797   B7A6
  LB79F   B795
  LB7B9   B7CD
> LB7C2   B7B0
  LB7C9   B7D9 B7DD B7E4 B812
  LB7E0   B7F7
  LB7E2   B7D3
  LB7EA   B7CF
  LB7F1   B7EB
  LB7F3   B7F0
  LB801   B808
  LB804   B80A
  LB80A   B802 B819
> LB814   B7E2 B80C
  LB820   B7CD B818
> LB821   AC9A ACAD
  LB829   B85B
  LB82D   B83A
  LB842   B83C
  LB844   B833
  LB852   B82F B838 B840 B846 B871 B880 B884 B890
  LB85A   B8C4 B802
  LB85C   B854
  LB86B   B860
  LB873   B86D
  LB87C   B8D9
  LB87E   B888
  LB880   B84C
  LB88A   B875
  LB892   B88C
  LB89B   B8D9
  LB89D   B8E8
  LB8AA   B8F5
  LB8AB   B8AC
  LB8C2   B8BA
  LB8C6   B9BE
  LB8CE   B8CA
  LB8D2   B8F4 B8E8
  LB8D4   B8A1
  LB8EA   B8B0
  LB8F1   B8F3
> LB8FE   B8F9
  L 907   B900
  LB911   B905
  LB91B   B909
  LB91B   B947 B999
  LB91D   B956
  LB93B   B930
  LB949   B941
  LB954   B94A
> LB95B   A3CF B8F7 B913 B943 B93F B963 B971
> LB95C   AC57 AC73 B7B0
  LB965   B91B B9A4
  LB966   B923 B950
  LB975   B9A8
  LB977   B96F B979
  LB97E   B91F
  LB98E   B97C
  LB992   B995
  LB997   B927 B973 B98C B990
> LB99C   A0DA A4F2 A93C AC68 AC79 AFED B0E4 B0D6
> LB99F   B91B B93B
  LB9A3    9AA
> LB9AC   B932 B7A8 B952 B992
> LB9AF   AC5A B02F B064
> LB9B1   ACA2 B78D B95A B9AB
> LB9B4   AE1F BF9D
> LB9B9   BF94
> LB9BC   BFBE
> LB9C2   B119 B9B7 B9A2 BF14 BF35 BFAC
  LB9C3   B9C0
> LB9CD   BB75
  LB9E2   B9D6
  LB9EC   BA42
  LB9FB   B9EE
  LBA1B   BC90 B8D6
  LBA1C   BA1B BAFD   03 BF71
  LBA1D   BA37
  LBA39   B719 BA59 B863
  LBA3E   B6D0
  LBA3F   B9D4 B9E4
  LBA44   BA4F
  LBA4F   BA1F
  LBA5C   BA16
  LBA66   BA76
  LBA72   BA64
  LBA78   BA72
> LBA79   BA1A
> LBA7B   BCD4
> LBA83   BA74
  LBA91   BA89
  LBA92   BA68 B867
  LBA97   B960
  LBA9A   BABO
> LBAAE   BA3F BCDE
  LBAB8   BAC2
> LBABA   BFE9 BCE8
  LBAC4   BA96
  LBAC9   AD7F BF32
> LBACA   BDFA BEFC
> LBAD0   BF4F
> LBB00   BADC BAE4 BAEC
> LBB02   BAF4
  LBB03   BB2C
  LBB20   BB08
  LBB2E   BACC BB06
> LBB2F   B9B9 B9C2 BACA
> LBB4B   BACE BB95
  LBB61   BB49 BB4F
  LBB63   BB55 BB5F
  LBB67   BB71 BB7A BB99
> LBB6A   BD7B BD8E BE11
  LBB7C   BB6D
  LBB7D   BB85
> LBB82   BDAF BE1B
> LBB89   BFB0
  LBBA4   BBDA
  LBB9D   BBAB BBAB BBB4 BBBA BBD8 BBDC
  LBBCC   B8C0 B8FA
  LBBD0   B8F6
  LBBDE   B8CE
  LBBF8   B8C8
  LBBFC   B8C6
  LBC06   BB91
> LBC0B   B4FA BC81
> LBC14   AD82 B110 B2B0 B8B8
> LBC2A   BF01
> LBC2F   B094 BEF2 BF2B
> LBC33   AFA0 B89B B11C
  LBC35   BC20
> LBC4A   B2DD B9C6
> LBC5F   B06A BBB2 BF7B BFD3
> LBC6D   AD9B BC7A BC9B BF1F
  LBC71   BC9E
  LBC73   BCC6
  LBC76   BC6F BC76
> LBC7C   B348 BD9C
  LBC82   B4FA
> LBC8B   BDD1
> LBC9A   B121 B304 B3F6
> LBCA0   BE04 BE0C
  LBCC3   B A4 BCAE BCB4 BCBA BCC0
> LBCCH   B3FB B74A BCF4 BE22
  LBCD7   BCDC
  LBCE4   BCDC
  LBCEE   BF38
  LBD09   BCCA
  LBD11   BCF2 BD81
> LBD12   B498 B22C B72A
  LBD2D   BD27
  LBD31   BD2B BD63 BD97
  LBD35   BD2F
  LBD53   BD43 BD47
  LBD55   BD4B BD4F BDB4
  LBD59   BD51
  LBD61   BD37
  LBD65   BD3B BD5B BD5F
  LBD6F   BD74
  LBD7B   BD6D BD7D
  LBD7F   BD6B BD76
  LBD86   BD20 BD33
> LBD89   BD95
  LBDA5   B03F B057
  LBD8A   BE09
  LBDBB   BE01
  LBDC0   B0F7
> LBDC5   AC7D
> LBDCC   B7AB
> LBDDA   BDC8
> LBDD6   B932 BDD4
> LBDDC   B503
  LBDE4   BDE0
  LBDFF   B0F5
  LBE01   BE1D
  LBE09   BE16
  LBE1B   BE07
  LBE1F   BE0F
  LBE36   BE2B BE2F
  LBE4B   BE3C BE45
  LBE50   BE6B BEBA
  LBE72   BE6D
  LBF04   BE7E
  LBEBC   BE90
  LBE9B   BE94
  LBEA3   BE9E
  LBEAB   BEAC
  LBFBR   BDEE
  LBEBA   BE9C
  LBEBC   BEB6
  LBEC0   B9B4
  LBFC5   BE4B
> LBEE9   B781 BD83 BE87 BFA6 BFB4
  LBBEF   BEEB
  LBEF0   BF8A
> LBEFC   BE95 BF0C BF30
  LBF01   BEF7
  LBF0C   BF1C
> LBF3B   BF26 BF8A
  LBF61   BEF7
  LBF0C   BF1C
  LBF3B   BF26 BF84
  LBF5B   BF24 BF2B
  LBF45   BF22
  LBF74   BF45

  LBF76   BF4A
  LBFA6   BF9B
  LBFA9   BFA2
  LBFB7   BFB2
  LBFBD   BF7B
  LBFC2   BF91 BF49
  LBFC7   BFB7
  LC000   A10A
  LC60B   A397 B449
  LC60E   AED1
  LC624   A61B
  LC626   A61E
  LC628   A618 B269
  LC62C   B26C
  LE060   A927
  LE8E4   A9FE
  LF0B6   A1DB
  LFF00   A1CB A9F7
  LFF01   A9A2
  LFF02   A9B8

LFF03   A936 A95B A9B3
LFF20   A02A A2FD A76E AB30 AB43 AB50 A987 A9F2
LFF21   A7CA A7EB A7F0
LFF22   A2ED
LFF23   A014 A0F6 A97A A981
LFFC0   A051
LFFC9   A05B
LFFFF   A63C AC7F AE17 B784
```

SUPPORT YOUR ADVERTISERS

** QUALITY SOFTWARE NEEDED **
Standard S50 Bus and Color Computer

For the past few months we at the DATA-COMP Division of Computer Publishing, Inc. (CPI), the parent company of 68 MICRO JOURNAL, have debated expanding into the software distribution business. Many other magazines have been doing so for years. Presently there are many fine examples of software that has been developed by YOU our readers, that will never see the 'light of day' unless someone, with enough exposure and willingness to continually advertise, runs with the ball.

Software is the 'backbone' for the real utilization of any computer, ours are no exceptions! Realizing that there will be some conflicts, with other advertisers, this has been no simple decision. However, since day one the foremost concern of 68 MICRO JOURNAL has been it's readers! Therefore, DATA-COMP Division will accept, for appraisal, software that runs on 6809 systems, games, utility or applications programs.

In the past there has been too much software offered that was not quite ready, nearly, but not quite. We will strive to eliminate that element. But right up front we tell you only that we will do our very best, nothing more. Also we will strive to keep cost to a bare minimum, while securing for the author a fair return, in royalty payments, promptly paid.

Of course we will expect, no - demand, that the author keep the product free of errors (bugs), and maintain it on a prompt and business like basis. Also we shall require that authors be willing to furnish 'source' for those programs that justify, by price and utility, inclusion of same. The lack of source code, properly commented, is a continual complaint we hear. Not all programs will be sold with source, but where necessary, we will insist that it be included.

In some instances the program may be small or short and not justify itself as a 'single' sale product. In this event it will be combined with other like programs, and offered as a package. In that event the royalties will be split between the various authors.

If you have software that you feel will qualify under this program please contact the proper person as shown below.

Standard S50 Bus
Don Williams
Bob Nay

Color Computer
Tom Williams
Bob Nay

Remember, if your software has any problems or 'funnies' - GET IT STRAIGHT BEFORE YOU CONTACT US!!!!! Also get your source code in proper shape and well commented. There is too much 99% code already drifting around.

DATA-COMP, POB 794, Hixson, TN 37343 - (615) 842-4601
A Division of CPI

"C" User Notes

Norm Cosmo
3 Pryor Road
Natick, MA 01760

I had hoped to start getting into the coding, in C, of some of the more interesting "standard" functions as way of exposing you to more code. Instead, I want to get out a quick review of the introl compiler, version 1.4 and reprint the file persusing program VU.C that got clobbered by the proportional printer in last September's issue.

INTROL VERSION 1.4

Version 1.4 is the latest release of the introl compiler. It has a number of improvements and now supports the float and long data types. First the improvements.

The "complier" is now composed of four programs. The first is the preprocessor, which has been named CP. It does both macro substitutions and lexical analysis and produces a file of strings and a file of lexemes. Next in line is C1, the parser. It takes the lexeme file and and produces a parse tree file and a symbol table file. Then comes C2, the optimizer. This is one of the major changes from earlier versions. The optimizer takes the parse tree and produces an optimized parse tree. Finally the code generator, C4, takes the optimized parse tree, string file and symbol table and produces the assembler code for the 6809.

This new hierarchy gives introl the ability to quickly build compilers for different machines by changing the code generator and recompiling the compiler. It's a little more complicated than that but not much.

While the compiler will chain to each pass automatically, including the assembler, it is not necessary to do so. Introl has included a number of compile time switches that give the user more control over the compilation process. These include the ability to specify additional directories for the compiler to search when looking for include files, where to find additional compiler passes, where to build temporary files, and whether or not to chain to the next phase.

Any phase can be started with the simple command syntax

C? <filename> <switches>

The filename does not need to include the extension. If the a particular phase being chained to isn't found on the disk then control reverts back to the operating system. In the case of OS-9, a 216 error is generated. This now allows users with 35 track single density minidisks to run the compiler by swapping disks. Admittedly, this is far from optimal, but better than not using the compiler at all.

As I mentioned earlier, the compiler now handles the data types float and long. The long is 32 bit signed integer with a range of -2147483648 to 2147483647. Unsigned longs are not supported. The floats are also 32 bits with a range of about 1e 38 to 1e38 with six to seven digits of accuracy.

The runtime package presently supports only add, subtract, multiply and divide. However, introl describes their internal format so that you should be able to interface it to your own higher order math routines with little trouble. Expressions with mixed data types that yield a float follows the conversion rules laid out in "The C Programming Language". In cases where a float converted to a smaller data type, the conversion is done by truncation.

The library reflects the additional data types with the inclusion of a few new functions and a rather complete version of printf(), which supports precision and other field formatting. They have also included an error handler for handling things like infinity and "not a number". I won't go into those at this time.

As of this writing, I have put a couple of programs, including vu.c, through the compiler with no trouble what so ever. I also used the librarian to add a function to the standard library. I may have run into a bug. I tried to install a small function, called hatoi(), behind exit() which was the second function in the library. The librarian squawked about hatio() being "too big". This seemed rather odd, so I then tried to insert if further into the library, but ahead of any other functions that it referenced. No problem this time, all worked as it should have. I have notified introl of the problem.

I have not tested the float and long data types other than displaying them with printf(). They printed out properly. I hope to have an update for you in a later column. Incidently, a reminder. C compilers expect that functions will normally be returning an integer (or equivalent). If a function returns something other than an integer, char or pointer, then you must declare what it does return so that the compiler can generate the correct code. If you write a function that returns a float or a long, then you have to declare that fact to the compiler.

When you are writing (declaring) the function it is done like this

```
float sin (angle)
    float angle;
    {
    .
    .
    .
    }
```

This tells the compiler that sin() will be returning a float.

When you want to use sin() in a program, you must again tell the c piler with the syntax

```
float sin();
```

I have found it easiest to declare all these "special" functions in a header file that I include with the various program modules. Even if the function is declared in the same file, you should declare what it is to any other function that must use it, either globally or within the calling function.

VU.C

Included in this month's column is a program for perusing a file. the complete description of how to use it was given in last September's column and will not be included here. The listing however, was mauled by the proportional printer which imparted its own special meaning to certain punctuation used rather heavily in C programs. I have compiled and run versions of this program with both the Words Worth and introl compliers.

It should be pointed out that the function dumpline() was written in a somewhat awkward fashion to avoid using the file string function fgets(), which was not implemented in Words Worth's FLEX compiler. If you are going to use the program on any other compiler, you might have to play some games with this function in terms of how it interprets the end of line character. Different run time packages may convert the carriage return, '\r', to a newline, '\n', when inputting from a file.

The version shown if for the Words Worth compiler. If you have the introl compiler then you should make the following changes.

1) Replace all the define statements with the following.

```
#include  "stdio.h"
#define CRTSIZE 24
#define BIGBUFF 265
```

2) Delete the function atoi() at the tail end of the program. It already exists in stdlib.lib

3) Use the following version of the dumpline().

```
dumpline(rmode)
    BOOL rmode;
    {
    char buff[BIGBUFF];

    if (fgets(buff,sizeof(buff),fcb) == NULL)
        return(FALSE);
```

```
    if (pmode)
        puts(buff);
    return(TRUE);
    }
```

I got a big surprise when I tried to run RLOAD for the Words Worth version. It couldn't find a getchar(). It turned out that Words Worth doesn't include getchar() as part of their library. So I wrote a very simple one and included it in cchar.lib. To put it in yours you will need to edit the files CEQUATES.TXT and CCHAR.ASM as follows.

In CEQUATES.TXT, include an equate for FLEX's getchr routine at $CD15.

In CCHAR.ASM, you will need to declare the entry point to getchar() and then include the code for it. Do this by inserting

```
   ENT getcha
```

before all the other ENT definitions. Next, insert the code for the routine before the code for gets().

```
*
* getcha     get a character from stdin
*
* ON EXIT
*
*   D - contains the input character
*
getcha  EQU     *
        JSR     GETCHR
        TFR     A,B
        SEX
        RTS
```

Here's a little hint to get you thinking. You can, to a degree, determine the final size of your program by what functions you use. For example, if you are not printing numbers try to avoid printf() like the plague. Call puts() instead. The following two code segments produce the same output on the terminal.

```
    printf("this is a string: %s\n\n",str);
```

or

```
    puts("this is a string: ");
    puts(str);
    puts("\n\n");
```

Why would you want to do it that way? Because a complete version of printf() is a BIG function. As a test, I compiled and linked the following two programs with the introl compiler under OS-9.

```
main()
    {
    puts();
    }
```

```
main()
    {
    printf();
    }
```

The version with puts() came out to 1182 bytes, whereas the version with printf() was 6863 bytes long. I used the librarian to determine the size of the runtime setup and exit code which was 179 bytes. That means that printf() is over 6K of code. You certainly wouldn't want to use it in a utility that should load in from the disk quickly. In the case of FLEX paying attention to this sort of thing may even mean the difference of whether or not a program can fit in the utility area at $C100.

WHAT'S NEW

I had talk with Microware recently and was glad to here that they are coming out soon with what they believe to be a very good C compiler. In its initial release, it will include all of C with the exception of bit fields. Initially, the whole package will include the compiler, relocating assembler linker, standard library and the source code to some of the standard functions.

Expect it sometime after March or April. It should be in the $400 price class. At a later date the assembler and linker will be sold separately.

That's a wrap for this month. A recent journal put out by the ACM contained some interesting Pascal procedures and functions for handling strings. I hope to have them coded up and tested in C for you in the next column. Till then...

```
/*
 * vu.c          rev: 1
 * n f combo
 *
 * last edit:    Feb-7-83
 *
 * A program to aid perusing disk files.
 *
 */
#define CRTSIZE 24
#define FOREVER while(1)
#define BOOL int
#define METACHAR int
#define TRUE 1
#define FALSE 0
#define ERROR -1
#define EOF -1
#define FILE char


FILE *fcb;

main(arec,argv)
    int arec;
    char *argv[];
    {
    int i;

    if (arec < 2)
        {
        printf("\nUsage:  vu are [args]");
        exit();
        }

    i = 0;
    while(++i < arec)
        {
        if ((fcb = fopen(argv[i],"R")) == ERROR)
            printf("\nError opening file: %s",argv[i]);
        else
            {
            display();
            fclose(fcb);
            }
        }
    }

display()
    {
    int linecnt, maxcnt, newmax;
    char c, ansie[10];

    maxcnt = newmax = CRTSIZE;
    linecnt = 1;
    FOREVER
        {
        while (linecnt++ < maxcnt)
            {
            if (dumpline(TRUE) == FALSE)
                return;
            }
        prompt();
        maxcnt = newmax;
        linecnt = 1;
        c = answer(ansie);
        if (isdigit(ansie[0]))
            maxcnt = atoi(ansie);
        switch(c)
            {
            case ' ' :  break;
            case 'f' :  killprompt();
                        printf("skipping %d screenfulls",maxcnt);
                        maxcnt = maxcnt * newmax;
                        skiplines(maxcnt);
                        maxcnt = newmax;
                        break;
            case 'q' :  killprompt();
                        return;
            case 's' :  killprompt();
                        printf("skipping %d line ",maxcnt);
                        skiplines(maxcnt);
                        maxcnt = newmax;
                        break;
            case 'z' :  newmax = maxcnt;
                        break;
            case '\r' : linecnt = maxcnt - 1;
                        break;
```

```
            case '\004' :   maxcnt = 11;
                            break;
            default :   printf("\007");
                        break;
            }
        killprompt();
        }
    }

skiplines(lines)
    int lines;
    {
    while (--lines)
        {
        if (dumpline(FALSE) == FALSE)
            return;
        }
    }

dumpline(pmode)
    BOOL pmode;
    {
    METACHAR c;

    FOREVER
        {
        if ((c = getc(fcb)) == EOF)
            return(FALSE);
        if (pmode == TRUE)
            {
            putchar(c);
            if (c == '\r')
                putchar('\n');
            }
        if (c == '\n')
            return(TRUE);
        }
    }

answer(s)
    char *s;
    {
    char c, *p;

    p = s;
    FOREVER
        {
        c = watchar();
        if (isdigit(c))
            {
            *p++ = c;
            continue;
            }
        else if (c == '\b')
                {
                if (p > s)
                    {
                    printf(" \b");
                    p--;
                    }
                else
                    printf(" ");
                continue;
                }
        else
                {
                *p = '\0';
                return(tolower(c));
                }
/*
 * the following two routines must be
 * change for the particular terminal
 * in use.
 *
 *
 *
 * 1) set inverse video
 * 2) write the prompt
 * 3) restore normal video
 */

prompt()
    {
    printf("\033p--more--\033q  ");
    }


/*
 * 1) set cursor at beginning of line
 * 2) erase the line
 */

killprompt()
    {
    printf("\r\033l");
    }


/*
 * an extremely simple atoi
 */
atoi(s)
    char *s;
    {
    int n;

    n = 0;
    while(isdigit(*s))
        n = (n * 10) + (*s++ - '0');
    return(n);
    }
```

OS9 USER NOTES

By: Peter Dibble
517 Golar House
Rochester, NY 14620

Editor's Note: This is the first of a new OS9 column. It is intended as both a tutorial and platform for discussion. Your response to the author is invited.

So we all here at 68 MICRO JOURNAL say - "Welcome aboard, Peter!"

* * *

This is the first of what I hope will be a long series of columns about OS-9 Level Two. I plan on discussing some interesting aspect of programming in each column. I also intend to use this as a soap box for my radical ideas about computing in hopes of stirring up some controversy. My computer was financed largely by teaching computer science. Please bear with me when I have a fit of teaching.

First, by way of introduction, I work as a systems programmer on a variety of machines. I teach computer science courses at a local technical college, and take computer science courses at the local university. I worked my way up to my job as a systems programmer through years of work on payroll, student systems, and other business programming type things - you might say I paid my dues. I got started on microcomputers by building SWTPC's 6809 computer kit. I now own two computers, one running only FLEX, the other large and frequently used. My large computer has a GIMIX DMA disk controller, and a GIMIX 6809 CPU board, two eight inch disk drives, 344K of useful memory, and assorted I/O boards. It can run OS-9 Level Two or FLEX.

I have a collection of strong opinions about computing in general, and microcomputing in particular. The most relevant opinion is that I think the staggering sum I spent to buy OS-9 Level Two together with languages and utilities was money that could not have been better spent though I do wish the prices were lower. I think everyone should get to watch their computer seem to come alive, not just those people who are willing to work two jobs and live on pasta to save enough money. I belong to the school of radicals who believe that Basic is bad for your brain. I like Pascal, but find it a little dull. Assembly language is lots of fun, but slow going. I am looking forward to getting C; it sounds promising.

I think it is practically immoral to force even two people to attempt to use a 6809 at the same time. The fact that it sometimes does a passable job with several users is not a sign that there is plenty of power for several users. The 6809 only runs just so fast. No operating system can make it run faster. Digital Equipment Corp. seems to think its small VAX is probably a good single user machine. I have noticed that the Xerox Star is very slow when it's editing. Both of those computers can run circles around any 6809 machine (and cost far more). Both of them run software written by top quality programmers. The difference is that those computers are expected to make things as easy as possible for their users at any reasonable expense. People who use and program microcomputers don't expect that much out of their machines. Our machines are microcomputers. We expect them to do the same kinds of things other microcomputers do. Our machines are small, but they are part of a new generation. They can do the work of several of last generation's micros. We can use that power to give several users the same poor service, but I would rather see one user well pleased by a computer than several somewhat dissatisfied users.

There is some truly excellent software available for the 6809. I would rate Microware's Pascal as one of the best Pascals I have used on any machine. A lot of features are missing from OS-9 Level Two, but what is there is up to the highest standards and it should be easy to add most of what's missing. From my reading of the manual, Basic09 seems to be an excellent language (as Basic goes). I own a great deal of software for FLEX and OS-9, but I can't think of any other programs in that league. I am open to suggestions. The challenge is to be at least as good as any similar program on ANY MACHINE. For example, I would love to find an editor that qualifies. My life would be much easier if I could run an editor comparable to EMACS, XEDIT, SPF, or SED on my micro. OYNACALC seems, from its advertisements, to be as good as any of the Visiclones, maybe the best of them. I am holding a grudge against that program because it only supports 3000 cells (under OS-9 Level Two) That's as good as most Visiclones, but I have enough memory for much more than that. The chance to be the first spread sheet program to support almost a megabyte of storage (maybe 30000 cells) in memory on a micro was only a few hundred instructions away, and they didn't do it. I would like to propose some programming challenges to the 6809 community.

I have used spelling checkers that can be asked for a list of suggestions for the spelling of a questionable word. The good ones will provide synonyms on demand too. Doing this at a decent clip, and fitting the dictionary on a floppy disk should be an interesting challenge, running OS9.

I don't know of any high level language for the 6809 that can use more than 64K even with restrictions. No, I take that back. Microware's Pascal can use a sort of virtual storage scheme to deal with more than 64K of code, but there is no easy way to use more than 64K of data. What I had in mind was a language that could make use of extended addressing. There are lots of useful tricks, playing with the DAT, using software interrupts cleverly, or simply running all procedures (subroutines if you like that term better) as FORKed tasks. Minicomputers used to be limited to a 64K address space. Some of the tricks used to fit big programs into them can probably be adapted to our problems.

A state-of-the-art editor would go a long way toward promoting the 6809. As basic requirements, such an editor should be a screen editor capable of using any available memory. It should include the ability to edit multiple files of arbitrary size without resorting to the "new" or "more" kludge. It should include the best of Wylbur, EMACS, and the other common editors.

If I seem a little shrill about software, it is because I see my beloved 6809 machine being squeezed out by the flood of high quality microcomputers on the market. From my point of view, the best feature of the 6809 is its elegant architecture. It is so easy to program that it should be pulling ahead of the field with a flood of superb software. I see only a trickle.

Ok, now I'll get off the soap box and down to business. I am thinking of selling both my computers. I positively lust after the new GIMIX CPU board and "level Three" operating system. If there is another microcomputer on the market that does what it does, I haven't heard of it. Large computers such as IBM 370 architecture, and large DECs can cause attempts to write into "protected storage" or execute invalid instructions to fail. Special code is executed whenever a program attempts either of these activities. Usually the program that did it is stopped. Microcomputers don't do that kind of thing. The computer will do something (maybe something ridiculous such as "halt and catch fire") with any data its program counter is pointed at. This can cause a faulty program to go out of control in unpredictable ways. There is no way for the microprocessor to know that it shouldn't write into some part of memory. If you want you can write your name all over the Basic09 interpreter. The results of that kind of thing are disastrous, particularly if you are sharing the interpreter with someone. You just have to make very sure programs you write never try to execute, or write into anything they shouldn't. Of course that is just good programming. The new board from GIMIX was designed to work with OS-9. It is alleged to support protected storage and to prevent invalid operations from being presented to the microprocessor. This should prevent any program from interfering with any other program, even, in many cases, itself. For those of you who try to support several users, if you use the new GIMIX hard/software no user should be able to cause the system, or another user's program to fail. Even people like me who don't share time with anyone can gain a lot from this kind of safety net. Sometimes when I am debugging a program everything just comes to a stop and I have to re-boot in order to continue. It is even worse when there is a long pause then the disk starts seeking. I haven't had any data destroyed that way yet, but I worry. This new hardware should give everyone who can afford it a lot of peace of mind. GIMIX has also been able to remove every trace of the operating system from each task's address space. Programs can be run with up to 64K. The board and accompanying software have lots of other features, but the other one that excites me a lot is the memory-to-memory DMA. A lot of time is spent moving data from one address space to another in OS-9 Level Two. This involves several operations for each byte and slows I/O operations and other inter-task communications down quite a lot. The special hardware on this new CPU board can move blocks of data at 2 cycles per byte. At two megahertz that comes to one million bytes per second. I understand that, all things taken together, the new system runs OS-9 substantially faster than what I have now. I want to find out for myself. If you see an advertisement from me in the classified section you will know I broke down and got a new, faster, better 6809 computer.

One of the nicest features of OS-9 (both levels) is the relative ease with which it can be adapted to new hardware. For example, there is a module included with the operating system called

ACIA which is responsible for interfacing the rest of the system with ACIAs (Asynchronous Communications Interface Controllers, or serial ports). There is another module called PIA which does a similar job for parallel ports, and another module which deals with whatever type of disk controller you have - more modules if you have more than one type of disk controller. If you feel the need you can add more Device Drivers (the name of this type of module) any time you like. If you want to write your own driver, it is good to have an example to work from. The source for ACIA and PIA (available from Microware) are both good starting places though I found ACIA more useful.

There is a rather odd sort of device which is available with most operating systems, but not OS-9. I have seen it called DUMMY and NULL. This device makes anything written to it disappear, and returns an endfile if it is read from. It is surprising how often it is nice to have any easy way to throw data away.

The Null Device Driver that I am going to present here is a SCF (Sequential Character File) type device. The requirements for this kind of driver are given in the OS-9 System Programmer's Manual, but in general there are six entry points: initialize device, read, write, get device status, set device status, and terminate the device. This driver is so simple that of those six, five just clear the carry bit and return. Read is the only operation requiring more than two lines of code. Read is supposed to return with the character read in accumulator A. If an error takes place, the carry bit should be turned on, and the error code placed in accumulator B. We want to return end-of-file, which is an error, and I have found that is a good idea to return null (Chr(0)) as the character read even if it is end-of-file. I return the end-of-file from the driver though it is usually generated by the SCF file manager. If you want to modify the program such that the file manager is the module that generates the end-of-file, load accumulator A with the end-file character which can be found in the path descriptor (pointed to by Y) and return with carry clear.

A Device Driver may be used for several devices provided that they use the same hardware. Each individual device is described by a "Device Descriptor" which includes everything unique to a particular device such as the address of the device. The NL device descriptor is at the bottom of the program. It will be loaded into memory at the same time as the Driver although it will show up as a separate module in the module directory.

Documentation for Null Device Descriptor

Summary: If the file Null is loaded and the module NL is linked
a new device called /NL will became available for input
and output.
OS9 Load Null
OS9 Link NL

The device NL will accept input in any quantity and simply
make it disappear. If a read is directed at it, it will
reply <end of file>. Other than eating data without a
sign it acts like a perfectly normal SCF type device...
a very fast and efficient one!

Example:
OS9: asm MyProg o 48k >/nl &
Would assemble MyProg in background and make all its (non-error
path) output disappear.

Note: Be careful when using /NL for input. Some programs (such as
debug) don't respond to <End of File> - these programs will
act very oddly if /NL is used as the input device for them.

```
Microware OS-9 Assembler 2.1   02/19/83 21:49:38                    Page 001
Dummy I/O driver - Definitions

00001                           NAM   Dummy I/O driver
00002                           TTL   Definitions
00003                   *-------------------------------------------------*
00004                   * Dummy                        1July82 Peter Dibble *
00005                   * return end of file to any read                    *
00006                   * Put any output down the bit bucket.               *
00007                   * No error returns                                  *
00008                   * Public Domain software as of 19Feb83.             *
00009                   *-------------------------------------------------*
00010                           IFP1                  Use OS9 definitions
00012                           ENDC                  but only on pass one
00013  00E1             Type    set   DRVR+OBJCT
00014  0082             Revs    set   REENT+2
00015  0000 87CD002E            MOD   Dummy1,DumNam,Type,Revs,Entry,Memsize
00016 D 0010                     ORG   V.SCF           leave space for SCFman overhea
00017 D 0010             Memsize equ   .
00018  0000 07                  fcb   READ.+WRITE.+EXEC. driver mode
00019                           TTL   Dummy I/O Driver
00020  000E 4460F9       DumNam  fcs   /Dmy/
00021  0011 01                  fcb   1               Edition number
00022  0012             Entry
00023 v 0012 160000F             lbra  Init
00024 v 0015 160000E             lbra  Read
00025 v 0018 160009              lbra  Write
00026 v 001B 160008              lbra  GetStat
00027 v 001E 160003              lbra  PutStat
00028 v 0021 160000              lbra  Term


00029  0024             Init
00030  0024             Write
00031  0024             GetStat
00032  0024             PutStat
00033  0024             Term
00034  0024 1F                  clrb                  zero return code
00035  0025 39                  rts                   Do nothing


00036  0026             Read
00037  0026 4F                  clra
00038  0027 53                  comb                  set carry flag
00039  0028 C803                ldb   #E$EOF          return eod of file
00040  002A 39                  rts                   return
00041  002B 848D39              emod
00042  002E             Dummy1  equ   *

Microware OS-9 Assembler 2.1   2/19/83 21:50:25                    Page 002
Dummy I/O driver - Dummy I/O Driver

00043                           TTL   Device Descriptor
00044                   *-------------------------------------------------*
00045                   * NL device descriptor                             *
00046                   *-------------------------------------------------*
00047  00F1             Type    set   DEVIC+OBJCT
00048  0000 87CD001E            mod   DDend,DDNam,Type,Revs,FMNam,DRVNam
00049  0000 07                  fcb   READ.+WRITE.+EXEC. modes
00050  0001 FF0000              fcb   $FF,0,0         PORT ADDRESS OF 0
00051  0011 0100                fcb   1,DT.SCF        Options
00052  0013 4ECC        DDNam   fcs   /NL/            device name
00053  0015 534358      FMNam   fcs   /SCF/           File Manager Name
00054  0018 4460F9      DRVNam  fcs   /Dmy/
00055  001B B05979              emod
00056  001E             DDend   equ   *

00000 error(s)
00000 warning(s)
$004C 00076 program bytes generated
$0000 00000 data bytes allocated
$20AA 08362 bytes used for symbols
```

COPYMULT - A disk backup system

If you have ever tried to backup a winchester to a smaller disk, or even an eight inch to 5 inch disk, you know what the word work means. Here is the typical procedure:

1. First you use the FLEX™ 'COPY' command to tell the system from where to where (drive ? to drive ?). By not calling by file name, the copy command will copy all files from source to destination disk. So far so good. Then the fan gets struck. You fill up the destination disk and the system stops and informs you that the disk is "OUT OF ALL AVAILABLE DISK SPACE." So you put another clean disk in the destination disk drive and start again, but wait... it starts all over again, at the beginning of the source. No good, already have those files on the disk we just filled up. So... the fun(?) begins.

2. Now you must, if you want to get down beyond the last file copied over completely, start the time consuming task of deleting all the copied files from the source disk. Sound like fun? Well, if your winchester is as full as some of ours are, you could well have over 3,000 file

names to delete, even on an eight inch disk you can have well over 200 file names to contend with. Not a happy prospect.

3. Now comes **COPYMULT** the hassle free large to small download program for FLEX™ (UniFLEX soon). You first run a directory of the source disk and determine about how many sectors you will be copying from one to the other. Next you 'NEWDISK' a sufficient number of destination disk to hold all the files. Now you **simply** call the **COPYMULT** command, just like FLEX COPY command, example:

```
+++ COPYMULT,2,1
```

Thats all there is to it.

COPYMULT keeps up with all files copied. As you near the end of disk space, **COPYMULT** figures if the next file ca be carried over in one piece. If not it stops, rings the bell, and informs you the next files requires X-number of sectors and only X-number are available on the destination disk. You many (A)bort - or (C)ontinue. If you continue you are directed to place the next clean disk in the drive and hit a key. And the process continues, **no deleteing, no keeping up with anything.** Just go about your normal duties and wait for the bell to ring. **WHAT COULD BE SIMPLER?**

COPYMULT even has a special program that takes all the risk out of copying files that are larger than the destination disk (i.e. data files, etc.).

Also included in the package is a program to relink the free chain on any disk. File access is greatly speeded up. This program **FREELINK.CMD** is included, at no extra charge.

As if all that is not the best bargain to ever come down the pike, since sliced bread, you get, **at no extra charge,** the source, well commented.

For those who have long fought this problem and do not desire to pop for the cost of other expensive backup systems, the price of **ONLY $99.95,** for the entire package is one sweet deal.

COPYMULT can be ordered from the following source:

DATA-COMP
POB 794
HIXSON, TN 37343
(615) 842-4601

P.S. It works on all FLEX™ systems, SWTPC™, GIMIX™, HELIX™, etc.

O-F * A PROGRAM TO TRANSFER OS9™ to FLEX™ & FLEX to OS9

As has been the case with other disk operating systems, the inability to transfer files from one to another, due to media differences, has plagued the microcomputer community, from day one. There are several excellent transfer programs available to the Standard S50 Bus user (Lucidata's COPYCAT, etc) as an example. However, the growing ranks of OS9 users has shown that media transfer programs are not just nice, they are **vital.**

O-F is a program that runs under OS9, in BASIC09, and from a menu performs the following functions:

1. DIR - this selection allows OS9 to run a directory of the disk, with all essential information.

2. Write a OS9 format file to the disk, to be read by a FLEX system.

3. Read a FLEX file on the disk and COPY or LIST to another OS9 device.

4. Format a 'soecial' disk that can be written to and read from by both OS9 and FLEX. This special disk is the 'go-between' disk that is used by both systems.

Also a special binary program is included to allow the transfer of binary files, with all the differences in system requirements accomplished.

FLEX users may use the special disk like any other FLEX disk, with one exception. The special disk while being used by FLEX must not allow the 'Directory' to overflow beyond track zero (too many files). Other than this one restriction, there is no difference between any other FLEX disk, as concerns the operator. However, if the disk is viewed, track by track. It will be found to be missing a sector on each track. This is the 'special' format and causes no problems, as long as the one restriction listed above, is compiled with.

For OS9 users the disk is more 'special' and some normal OS9 functions will invoke errors. The special disk should be used, by OS9 operators, as a transfer and program disk only. The 'special' disk must be formatted under OS9 using the BASIC09 programs furnished.

O-F is furnished on 5 or 8 inch diskettes, in SOURCE. The current price, in OS9 format is **$79.95.**

DATA-COMP
POB 794
Hixson, TN 37343
1-(615) 842-4601

- - -

FAST TRIG&MATH

HI SPEED TRIG AND MATH FUNCTIONS
USING POLYNOMIALS & THE CORDIC TECHNIQUE

by Matt Scudiere
100 Cedar Ln.
Oak Ridge, Tenn 37830

Editor's Note: All occurances of the 'up arrow' are indicated by the symbol '↑'.

POLYNOMIALS

It seems that in any system with floating point capability the need for powers, logarithms, and trig functions arise quite frequently. Calculating these functions can get to be a very time consuming occupation for the processor to say the least if the program under execution requires a lot of these function calls. The most common method used by many systems is to use a fitted polynomial to the desired function. Calculating polynomials is very straight forward and is easily encoded. One simply supplies the desired coefficients for the required function. The calculation for a polynomial:

P(Z) = A0 + A1*Z + A2* ↑2 + A3*Z↑3 + ... + AN* ↑N

can be broken down to:

P(Z) = A0 + Z*(A1 + Z*(A2 + Z*(A3 + ... Z*(AN)...)))

which reduces the number of multiplies and which can be encoded as:

```
P=AN
FOR I=N-1 TO 0 STEP -1
P=P*Z+A(I)
NEXT I
REM P = desired polynomial on exit.
```

This is a very efficient method since there are N multiplications and additions. In addition truncations errors are minimized. If one were to encode the first equation directly there would be (N+1)*N/2 multiplications and in many systems truncation errors could limit the number of terms to somewhere in the vicinity of six or seven depending on the range of the particular numbers involved. However, the number of terms needed are determined by the accuracy required in the polynomial approximation, the desired function, and the range of the function involved. POLYNOMIAL FITS ARE ONLY AN APPROXIMATION TO ANY FUNCTION THAT IS NOT ITSELF A POLYNOMIAL! To represent any nonpolynomial exactly would require an infinite number of terms. In the case of finite precision arithmetics in any computer the number of terms to "exactly" fit a function to the full precision is finite but can become very large. The exact number of terms depends on both the function being represented and the precision of the arithmatic being incorporated. One anomaly that occurs with polynomial fits that stop short of the precision being used is that the fitted polynomial oscillates about the function up to N times. If one is taking small differences of large numbers this oscillation distorts the answer to a greater extent than the original error. Usually this is not a problem but should be remembered if accurate small differences are needed. Coefficients for various functions with their respective errors can be found in many math table handbooks. Some of these least squares coefficients are listed in table I at the end of this article. So much for polynomials.

THE CORDIC TECHNIQUE

While polynomials are very easy to encode there exists another simple elegant algorithm based upon the CORDIC technique developed by Volder. The basic algorithm was first described by Henry Briggs in 1624 in Arithmetica Logarithmica. This algorithm which appears to be relatively unknown is easily implemented on any computer where one has access to "bit fiddling". Without the bit fiddling capability this algorithm does not offer much improvement in speed over the polynomial method. However, by accessing the exponent and mantissa of the floating point number separately one can eliminate most all of the floating point multiplications and divisions for several of the trigonometric functions and the LOG function. The exceptions to this statement are that several of the functions do require a square root calculation which normally requires M multiplications where M is the number of bits in the mantissa. With this one minor drawback we will proceed with the basis of the algorithm and what makes it so efficient.

THEORY BEHIND THE CORDIC TECHNIQUE

Assume that we want to calculate the TANgent of a given angle Z. We can break Z into two parts where Z = A + B. Then

$$TAN(A+B) = \frac{SIN(A+B)}{COS(A+B)} = \frac{SIN(A)COS(B) + COS(A)SIN(B)}{COS(A)COS(B) - SIN(A)SIN(B)}. \quad (1)$$

Dividing the numerator and denominator by COS(A) we get

$$TAN(A+B) = \frac{SIN(B) + TAN(A)COS(B)}{COS(B) - TAN(A)SIN(B)}. \quad (2)$$

Now, since we know the SIN(0) = 0 and COS(0) = 1 we can compute the tangent of any angle given a small lookup table of tangents. For example if our table were to consist of the tangents of 1, .1, .01, ... , .000001, and the tangent of .103 is required, then we can repeatedly apply this last equation computing the result as TAN((((0+.1)+.01)+.01)+.01). At first glance it looks like the sin and cos of the angles also needed to be calculated but since we can introduce an arbitrary multiplier C, which will cancel we will make the following definitions:

$$X = C * SIN(B) \quad \& \quad Y = C * COS(B).$$

Then

$$TAN(A+B) = \frac{X + Y*TAN(A)}{Y - X*TAN(A)} = \frac{SIN(A+B)*COS(A)}{COS(A+B)*COS(A)} \quad (3)$$

By inspection we see that X is proportional to SIN(A+B) and Y is proportional to COS(A+B). The arbitrary multiplier C we see is determined from inspection to be

$$C = COS(A).$$

Therefore X and Y can be update each time by

$$X = X(old) + Y(old)*TAN(A) \quad \&$$
$$Y = Y(old) - X(old)*TAN(A). \quad (4)$$

NOW HERE COMES THE KEY TO THE WHOLE PROCESS! If one is clever he will choose his table of tangents such that the angles represented for the summing will produce TANgents that are binary powers of two, so that the multiplications in equations (4) can be represented by arithmatic shifts if one is working in fixed point arithmetic or by subtracting a number from the exponent in a binary representation of the floating point number. The time it takes to do this operation is more than two orders of magnitude smaller than the time to perform the f.p. multiply. This process will work for any radix. It does not have to be base 2 but the efficiency is slightly higher for this than for say base 10 if one were using BCD. This radix must match the radix used in the f.p. representation being used. The table now will consist of an array ATAN(I) which stores only the angles since the tangent was picked to be 2↑(-I). If one stores ATAN(I) in terms of radians then the algorithm will produce results for angles in radians. However, for the trigonometric functions there is nothing sacred about these units for this algorithm. If one stored the angles in units of degrees, 256-ths of a circle, grads, etc. then the result will reflect the same units. The only caution here is that to compute the hyperbolic, exponential, and logrithmic functions, we do need the table in terms of radians if no conversions are wanted. The complete algorithm follows:

```
BEGIN /* find the SIN, COS, or TAN of angle A > 0 */
  /* initialize variables */
  X=0.0;
  Y=1.0;
  J=0;
  REPEAT
     A = A - ATAN(  -J); /* get ATAN from table ATAN(J) */
     IF A >= 0.0 THEN
        Xnew = Xold + (Yold *  -J); /* shift op here */
        Ynew = Yold - (Xold *  ↑-J);     "     "
     ELSE
        A = A + ATAN(2↑-J);
        J = J+1;
  UNTIL J > MAX;  /* MAX is the number of bits of
precision */
  SIN(A) = X / SQRT(X↑2+Y↑2);
  COS(A) = Y / SQRT(X↑2+Y↑2);   > pick which
  TAN(A) = X/Y;
  END;
```

One will see that in the case of the TAN there is only one divide and in the case of SIN & COS there is also a SQRT. If one normalizes his arguement to the range of 0 to P1/2 then the maximum number of times through the loop is MAX+2. Again note that there are no floating point multiplications or divisions within the loop.

One can follow the same process for the hyperbolic functions and obtain the following algorithm:

```
BEGIN /* find the SINH, COSH, or TANH of angle A */
/* interval should be restricted to 0 - 8 since
TANH(8+)=1.0 */
  /* initialize variables */
  X=0.0;
  Y=1.0;
  J=1; /* note: start at 1 for hyperbolic functions */
  REPEAT
     A = A - ATANH( ↑-J); /* get ATANH from table ATANH(J)
*/
     IF A >= 0.0 THEN
        Xnew = Xold + (Yold *2↑-J); /* shift op here */
        Ynew = Yold + (Xold *  -J);  " /* note "+" here */
     ELSE
        A = A + ATANH(  -J);
        J = J+1;
  UNTIL J > MAX;  /* MAX is the number of bits of
precision */
  SINH(A) = X / SQRT(X↑2-Y↑2); /* note: subtraction */
  COSH(A) = Y / SQRT(X↑2-Y↑2);    "      "
  TANH(A) = X/Y;
END;
```

Now most people really don't care about hyperbolic functions but you probably are interested in EXP(A). This can be calculated easily by

$$EXP(A) = SINH(A) + COSH(A) \quad \&$$
$$EXP(-A) = COSH(A) - SINH(A).$$

Or in terms of X & Y above

$$EXP(A) = (Y+X)/SQRT(Y↑2-X↑) \quad \&$$
$$EXP(-A) = (Y-X)/SQRT(Y↑-X↑). \quad (5)$$

The INVERSE functions can also be easily calculated in the same manner by simply reversing the process above.

```
BEGIN /* CALC ARCTAN, ARCSIN, or ARCCOS of value X > 0 */
  /* initialize variables */
  Y = 1.0;
  Z = 0.0;
  J = 0;
  REPEAT
     X = X - Y *(2↑-J);
     IF X >= 0.0 THEN
        Y = Y + ( X*2↑-J );  /* SHIFT OPERATION HERE */
        Z = Z + ATAN(  -J); /* same table as before */
     ELSE
        X = X + Y *(  -J);
        J = J+1;
  UNTIL J > MAX /* MAX = # bits of precision */
  /* Z NOW EQUALS ATAN(X) */
  ASIN(X) = Z / SQRT(1.0-Z↑2);   > pick which
  ACOS(X) = SQRT(1.0-Z↑2) / Z;
  /* if arg of acos upon entry is negative then add PI to
result of ACOS */
END.
```

Similarly the inverse hyperbolic functions have the following algorithm:

```
EGIN /* CALC ARCTANH of value X ( 0 <= X < 1 ) */
  /* initialize variables */
  Y = 1.0;
```

```
  Z = 0.0;
  J = 1; /* note: init to 1 for hyperbolic */
  REPEAT
     X = X - Y *(2↑-J); /* same table as before for Hyp */
     IF X >= 0.0 THEN
        Y = Y - ( X *2↑-J ); /* SHIFT OPERATION HERE */
        Z = A + ATANH(2↑-J);
     ELSE
        X = X + Y *(2↑-J);
        J = J+1;
  UNTIL J > MAX /* MAX = # bits of precision */
  /* Z NOW EQUALS ATANH(X) */
END.
```

Again you say "Pool who needs the inverse Hyperbolic functions?" Well, I'll bet you can use the LOG function. It can be calculated from the identity:

$$LOGe(X) = 2*ARCTANH[\ (X-1)\ /\ (X+1)\ ] \qquad (6)$$

And for what it's worth the other inverse hyperbolic functions can be computed from the identities:

$$ARCSINH(X) = LOGe[\ X\ /\ SQRT(\ X\uparrow 2 + 1\ )\ ],\ \&$$
$$ARCOSH(X) = LOGe[\ X\ /\ SQRT(\ X\uparrow 2 - 1\ )\ ]. \qquad (7)$$

SO FAR SO GOOD! WHAT ABOUT NORMALIZING?

The biggest headache when it comes to using any algorithm for these functions is that the algorithm is defined or works most efficiently only over a certain limited region. The trigonometric functions are easily scaled by symmetry to an interval 0 to PI/2 by successively subtracting (or adding) values of 2*PI, PI, & PI/2. The only thing to watch out for is to remember the correct sign and range for the result.

In the case of the hyperbolics, exponential, and logarithm functions the scaling process is a lot less obvious. By using some identities and noting the method of storing floating point numbers the process can be reduced to a extremely quick computational scheme. Let's look at the logarithm scaling first. The identity

$$LOGe[\ M * 2\uparrow N\ ] = LOGe(M) + N*LOGe(2)$$

shows us how this can be done. Here M is the mantissa of the number which is scaled to fall between 1 and 2 in the IEEE floating point format. Just right for our calculation using eq (5) above. The f.p. exponent stored in the number is N which has a 127 offset (bias) in the IEEE format. Therefore the scaling algorithm would be:

```
BEGIN /* scale X for call to LOG */
  separate X into M & N where M is the mantissa and N is
  LOG(M) = 2 * ARCTANH [ (M-1) / (M+1) ];
  LOG(X) = LOG(M) + (N-bias) * 0.693147181;
/* bias is the bias used in representing the f.p.
exponent */
END.
```

Next, to scale the exponent we can use the identity

$$EXP[\ N*LOGe(2) + R\ ] = 2\uparrow N * EXP(R),$$

where N & R (the remainder) are calculated by:

$$N = INT[\ X\ /\ LOGe(2)\ ];\ \&$$
$$R=X - N * LOGe(2); \qquad (8)$$

The reason for picking a radix of 2 is that the *2↑N operation can be performed by simply adding N to the exponent of the floating point representation. The algorithm then becomes

```
BEGIN /* calc EXP(X) where X may be large */
  N = INT(X*1.44269504); /* note: this needs to be
truncation */
  R = X - N; /* the remainder 0 <= R <= LOGe(2) */
  Y = EXP(R); /* use normal algorithm */
  EXP(N) = Y *2↑N;
/* this last op is adding N to the f.p. exponent of Y */
```

HERE'S THE TRICK

THE KEY TO THE EFFICIENCY OF THE CORDIC TECHNIQUE AND EXPONENTIAL SCALING IS BEING ABLE TO MULTIPLY NUMBERS BY 2↑N WITHOUT DOING A FLOATING POINT MULTIPLY!!! This is where the "bit fiddling" comes in. One needs to be able to split the floating point number into the normalized mantissa and the integer exponent (with or without the bias.) With these two numbers in hand one simply does an 8 bit integer add of N to the f.p. exponent to multiply the f.p. number by 2↑N and recombines the exponent with the normalized mantissa. If one is encoding this process in assembly then this is trivial. If, however, one is encoding this technique in a higher level language then one must be able to split the f.p. number without time wasting arithmetic operations or at least be able to call an assembly language routine that does this. The splitting of the number can usually be accomplished in the higher level languages by equivalencing an integer with a real number. One then accesses the exponent by masking the integer form of the number and shifting the appropriate number of spaces. THIS OPERATION IS DEPENDENT ON THE FORM OF THE FLOATING POINT NUMBER AND NEEDS TO BE TAILORED ACCORDINGLY!!!

A WORD ABOUT THE SQRT FUNCTION

Normally a SQRT function requires M-1 floating point multiplies where M is the number of bits in the mantissa. Since the numbers are normalized the first bit in the resultant mantissa is always a 1 and the remaining M-1 bits are determined by trial and error. Each bit in the trial is set in turn and the trial is squared and compared to the arguement. If the result is too large then the bit is reset. The process then continues with the next bit. There are some things that one can do to improve the situation though if one is willing to get down into the assembly code. For example: The f.p. exponent of the result is always the unbiased f.p. exponent of the arguement divided by two (8-bit ASR). If the arguement's exponent was odd (carry set) then its mantissa will need to be renormalized only if the mantissas are to be worked on directly. Next since we are only concerned about a 3 byte mantissa the full floating point operation does not need to be carried out on each iteration of the next bit. If one allows for the deletion of one or two possible carrys into the least significant bit then only 60 8*8 bit multiplies are necessary to calculate the square root of the number. The purist may want to include the possibility of the two carrys in which case there are twenty more 8*8 bit multiplies. For comparison a full floating point multiply implementation of the square root would require 216 8*8 but multiplies plus all of the overhead of the f.p. normalization. In any case the process can be trimmed to approximately one half to one fourth of the usual time if one is willing to put forth the effort. This would then make the CORDIC technique comparable in execution speed to the polynomial for all of the functions requiring the square root function. However, if precision to better than 1 part in 10↑9 is desired then the polynomial timings would increase slightly faster than the optomized SQRT/CORDIC technique. The question that remains is "Is it worth it?"

WHICH TECHNIQUE DO I USE?

First of all if you don't care about speed then use the polynomials for all your functions because they are so easy to encode. On the other hand if speed is of some importance then the Cordic technique will significantly improve the execution speed of some of these functions. The SQRT function is the only thorn in side of the CORDIC technique. It is used in the SIN, COS, ARCSIN, ARCCOS, SINH, COSH, ARCSINH, ARCCOSH and EXP. In these cases it is more efficient to use the polynomials. Those functions that do not need it are the TAN, ARCTAN, TANH, ARCTANH, and LOG. These are the functions that can realize a significant speed increase over the polynomials.

As a side note, on the 6809 one can easily deter ine whether or not the multiply instruction is being used for a numeric multiply. If it is then the multiply time will be on the order of a third of the divide time. If the rotate and shift is used then the multiply time will be just under the divide time.

TABLE I
LEAST SQUARES COEFFICIENTS FOR SOME COMMON FUNCTIONS
F(X) = SUM (AN * X↑N)
All coefficients not listed are zero.

function	(1/X)*SIN(X)	COS(X)	(1/X)*TAN(X)
arg range	0<=X<=PI/2	0<=X<=PI/2	0<=X<=PI/4
error	2*10↑-9 (29bits)	2*10↑-9 (29bits)	2*10↑-8 (25bits)
A0	1	1	1
A2	-.16666 66664	-.49999 99963	.33333 14036
A4	.00833 33315	.04166 66418	.13339 23995
A6	-.00019 84090	-.00138 88397	.05337 40803
A8	.00000 27562	.00002 47609	.02456 90893
A10	-.00000 00239	-.00000 02605	.00290 09250
A12			.00931 68091

function	LOGe(1+X)	EXP(X)-1	(1/X)*ARCTAN(X)
arg range	0<=X<=1	0<=X<=LOGe(2)	0<=X<=1
error	3*10T-8 (25bits)	2*10T-10 (32bits)	2*10T-8 (25bits)
A1	.99999 64329	.99999 99995	A0 1
A2	-.49987 41238	.49999 99206	A2 -.33333 14528
A3	.33179 90258	.16666 53019	A4 .19993 55085
A4	-.24073 38084	.04165 73475	A6 -.14208 89944
A5	.16765 40711	.00830 13598	A8 .10656 26393
A6	-.09532 93897	.00132 98820	A10 -.07528 96400
A7	.03608 84937	.00014 13161	A12 .04290 96138
A8	-.00645 35442		A14 -.01616 57367
			A16 .00286 62257

CORDIC TABLE DEFINITION

The table of inverse tangents and inverse hyperbolic tangents can either be calculated the first time a function is called or, more efficiently, calculated once and stored explicitly in the routines. The first eight terms are listed in table II. The rest of the terms can be generated with the first couple of terms of the following series:

$$\text{ARCTAN}(X) = \frac{X}{1} - \frac{X^3}{3} + \frac{X^5}{5} - \frac{X^7}{7} + \ldots \quad (X)<1$$

$$\text{ARCTANH}(X) = \frac{X}{1} + \frac{X^3}{3} + \frac{X^5}{5} + \frac{X^7}{7} + \ldots \quad (X)<1$$

Note:
arctan(x) = arctanh(x) = 2^-I to 2^-I bits precision &
arctan(x) = 2^-I * (1 - 1/3 * 2^-2I) to 4^I bits,
arctanh(x) = 2^-I * (1 + 1/3 * 2^-2I) to 4^I bits, etc.

By examining the series we see that they are both rapidly converging and that for 24 bits of precision and for I > 12 that the arc functions are both just 2^-I. Therefore the arrays are generated by calling our 2^-I function operating on 1.0 for all values of I between 12 and 24. For I between 6 and 12 we can include just the first term in the series and still have 24 bit precision. Including the third term would give a precision of 6*I bits, the fourth term would yield 8*I bits, etc. Therefore we can predetermine the number of terms required in the series given the desired precision.

TABLE II RADIAN ARCTANGENT TABLE FOR CORDIC TECHNIQUE

I	2^-I	ARCTAN(2^-I)	ARCTANH(2^-I)
0	1	0.78539 81633	Int (not used)
1	.5	0.46364 76090	0.54930 6144
2	.25	0.24497 86651	0.25541 2812
3	.125	0.12435 49945	0.12565 7214
4	.0625	0.06241 8810	0.06258 15713
5	.03125	3.12398 3343 E-2	3.12601 785 E-2
6	.015625	1.56237 2862 E-2	1.56262 7185 E-2
7	7.8125E-3	7.81234 1060 E-3	7.81265 870 E-2
8	3.90625E-3	3.90623 01 E-3	3.90626 986 E-3
9...	see discussion.		

THE COMPLETE C LISTING

For those who are interested a complete encoding of these functions in INTROL C is available on either OS9 or FLEX formatted disk (specify which) and can be obtained for $15 copy and disk charge (5" disk or $20 8" disk) from me at the above address. These functions as encoded also contain a few extra variations in some of the functions to increase precision in certain ranges while reducing computational time at the expense of a slightly larger runtime package.

END.

EXPERIMENTING WITH FLEX

Experimenting with Flex
by Peter A. Stark

I recently needed to quickly copy a sector of data from one disk to another. Rather than write a complex program, I decided to do it directly from the terminal. It makes a good experiment, as it will give you a better feeling for what goes on in your system. I will describe how it works on a 6809 system with Flex 9; adapting to other versions of Flex is trivial.

Let's suppose you want to read a single sector from the disk into memory. First boot your Flex system, and return to the monitor. Enter the following program directly into memory using your monitor:

```
1000  BE 2000    LDX #$2000    Point to FCB
1003  BD 0406    JSR FMS       Go to Flex FMS
1006  3F         SWI           Return to monitor
```

(This is written for 6809 Flex 9.0; for the 6800, use CE 2000 for the LDX instruction, and use B406 instead of D406. For MiniFlex, use 7806 instead of 0406.)

This program points to an FCB or File Control Block, which is used to tell Flex what you want to do and give it an area of memory to do it in, and then jumps to the FMS or File Management System entry point, so that Flex will execute the required function. After the function is completed, an SWI instruction returns to the monitor so we can examine what happened. (Note: the SWI is the same as a monitor breakpoint, and with some monitors it may be necessary to insert a breakpoint rather than put in a 3F instruction. Note also that Flex 2.0 erases the SWI pointer in memory, so that executing a breakpoint does not return to the monitor as you'd expect. To restore this pointer, do a jump to $E000, the starting point of the 6800 monitor.)

The above three-step program sets up the entry point into Flex, but before we can execute it we have to provide an FCB at address 2000. Do this by entering the following data:

```
2000  09
2003  00
201E  01
201F  02
```

The 09 at location 2000 is the FMS function code, and it tells Flex what we want to do. An 09 means "read a sector", while an 0A would mean "write a sector."

The 00 at location 2003 is the drive number. In this case we specified drive 0, but could also have entered 01, 02, or 03, depending on the number of drives on our system.

The two bytes at 201E and 201F specify the track and sector we want. In this case, I arbitrarily specified track 1, sector 2.

Now that everything is set up, place a spare Flex disk into drive 0 (just in case something goes wrong, we don't want to clobber your best disk) and jump to location 1000.

The motor will now start, and after a moment's delay the program will return to the monitor and give a register dump.

At the left of the displayed register dump is the contents of the condition code register. If it is displayed as a row of six or eight bits, the third bit from the right is the Z or Zero flag. If the sector was read correctly, this bit will be a 1. If it is a 0, then an error occurred. (If the condition code is displayed by your monitor as a two-digit hex number, then the second digit should be 4 through 7, or C through F, if no error occurred.)

Once the sector has been read, the 256 bytes in that sector will start at location 2040 in memory, and you may use your monitor to examine them. (MiniFlex will have only 128 bytes starting at 2040).

If you now experiment a bit, you will find that in Flex 2.0, you always get an error on reading sector 1 of track 0. That is because there is no sector 1 - the ten sectors on track 0 are numbered 0, 2, 3, 4, and so on, through 0A. In the same way, in MiniFlex you will always get an error when you try to read sector 2 of track 0. That's because MiniFlex numbers the sectors of track 0 as 0, 1, 3, 4, ... up to 0A.

On the other hand, all the other tracks have a sector 1, but lack sector 0. In other words, each track has 10 sectors, but they are numbered differently on track 0 from all other tracks. This is a peculiarity of 6800 Flex; in 6809 Flex all the tracks (including track 0) start with sector 1.

The reason has to do with the way the system is booted. The first SWTP disk system came with a disk operating system called MDOS. In order to start MDOS, a 'boot' or 'bootstrap' program was written

which would read the beginning of the disk (starting at track 0, sector 01 into memory, and then jump to the program just read in. This boot program was then put into the SWTBUG monitor.

When MiniFlex and Flex 2.0 were developed, the same SWTBUG boot program had to be used to load Flex as well. But Flex was too big to be loaded directly, so the boot was used to load a second boot program, sometimes called the 'superboot' from the disk. Once the superboot was loaded, the computer would jump to it. Now the superboot would load the rest of Flex from the disk. (The 6800 superboot loads in locations 2400 through 24FF ... thereby clobbering anything else that may be in those locations. Perhaps you'd noticed that.)

But the superboot program only required 256 bytes of memory, and so the problem came of how to stop further input once the superboot was in. The solution was simple - skip a sector so that the 1771 disk controller would quit on an error. Since MiniFlex had only 128 bytes per sector, it had sectors 0 and 1 and then sector 2 was missing; since Flex 2.0 used 256 bytes per sector, it had sector 0 and sector 1 was missing. (6809 Flex has a different boot program, and so its track 0 is numbered the same as all other tracks, starting with sector 1 and having no missing sectors.)

If you want to examine the disk a bit more, look at sector 3 of track 0. This is called the System Information Record or SIR, and holds quite a bit of information about the disk. As you examine memory, you will find the following:

Location 2050 - the disk name
205B - disk number
205D - the first sector in the free chain. This is the next sector that will be used if you write a new file to the disk.
205F - the last sector in the free chain. More on the word 'chain' in a moment.
2061 - the number of free sectors
2063 - date of disk creation - month
2064 - day
2065 - year
2066 - last track and sector on the disk.

(MiniFlex does not have a disk name or number, and the SIR is arranged a bit differently.)

After the SIR sector, Flex 2.0 and Flex09 leave sector 4 empty, and the directory starts in sector 5 (MiniFlex's directory starts in sector 4.) As you can see, there is a lot of empty space at the beginning of a disk, and an ambitious programmer could play some games here. For example, commercial software could be supplied with data in sector 4 of track 0. The program could then check for this data and run if the data were found, but quit (or make slight errors) if the data were missing. Since copying a disk does not preserve the data on track 0, sector 4, the program would run from the original disk but not from a copy. Obviously a MIRROR or BACKUP utility would defeat that, though.

Now let's look at other sectors.

Flex 'chains' sectors together, so that each sector points to the next one to be used. The first sector of each file is listed in the disk directory after the file name (and the first sector in the 'chain of free sectors' is listed in the SIR). Each sector in the file (or in the free chain) then points to the next sector, and finally the last sector points back to track 0 sector 0; this signifies the end of that chain.

To see how the pointing is done, read sector 5 of track 0. This happens to be part of the disk directory. You will see that the first two bytes (locations 2040 and 2041) of the data are 00 06. Hence track 00 sector 05 points to track 00 sector 06. Each sector in turn points to the next one in that chain. (The next two bytes, at 2042 and 2043, are a counter which indicates how many sectors came before the current sector in this chain. Directory sectors are not numbered, but file sectors are.)

On a freshly formatted disk, every sector on the disk (with the exception of some of those on track 0, and with the exception of the very last sector on the disk) simply points to the next sector, since the entire disk is one big 'free chain'. As a disk is used, that free chain is gradually shortened, and eventually sectors may point all over the disk as files become fragmented. (If you have the TSC INTEG utility, you can see that its printout follows the free sector chain on the disk and finally ends with a printout of 00 00, pointing to track 0 sector 0.)

How do you write a sector? First set up the data you want to write in addresses 2040 and up, then change location 2000 from 09 (for read) to 0A (for write), and again jump to location 1000. The data will be written into the sector specified by locations 2003, 201E, and 201F.

If you wanted to change the data in a sector, simply read the sector with an 09 in location 2000, then change a byte of data (which starts at location 2040), change location 2000 from 09 to 0A, and jump to 1000 to write the sector back.

As you can see, accessing a particular disk sector or even changing it is a cinch in Flex.

- - -

PASCAL-FLEX MODULES

BY SAM DEAN
ENERGY MANAGEMENT CORP.
19208 VERMONT ST.
GARDENA, CA 90248

HOW TO CHANGE GLOBALS OF PASCAL MODULES USING FLEX UTILITIES

When a program becomes large and unwieldy, we divide the program into separate modules. Each module contains one or more procedures. Each procedure executes a particular function. When programming in PASCAL, the outermost block (ie. the PROGRAM level) in these modules contain no instructions. But the outermost block does contain the global variables. One's application program will run properly only if the global variables in every module is the same and is in the same order. In order to change globals, each module must be correctly edited, compiled and assembled. If one of these steps is omitted or done incorrectly on one of the modules, one's program will not run properly. After four or more modules have been created, changing globals becomes time consuming and fraught with errors. To solve this problem, I have created an automatic global changer using the FLEX EXTRACT utility. This global changer was very useful when it was first created. Now it has become indispensible as our main program has grown to encompass more than 50 modules.

The first step in creating an automatic global changer is to add a comment at the top and at the bottom of the global section which contains a marker. We use the Omegasoft compiler which uses the parenthesis and the asterisk to enclose comments: (* comment *). At the top of each global section I added the comment:

(*> Global variables in the satellite. *1

And at the end of the global section I added the comment:

(* End of global section. <*)

The top of the global section marker is '(*>' and bottom of the global section
marker is '<*)'. Then I created a program called EX CMD which searches the text file for these markers to create a command file used by the EXTRACT utility. The EXTRACT utility creates a file from segments of other files. It is not necessary to copy the segments to scratch files and concatenate them. A command file is used to tell EXTRACT which lines are to be read from the files and concatenated to form the new file. In this application the top section above the top marker is saved and bottom section below the bottom marker is

saved and the new global section is inserted in the middle. For instance if a file named RCDATA has a global section which starts on line 31 and ends on line 69, then EX CMD will create a file called TEMP with the following three directives:

```
)RCDATA -31
)TEMPLATE
)RCDATA 69-
```

The new global section is contained in the file called TEMPLATE. To execute EXTRACT two file names are entered on the command line. For example

```
EXTRACT NEW,TEMP
```

creates a new file called NEW and uses the command file described above, TEMP, to create it. All that is left to complete the global changing process is to delete the old file, RCDATA, and rename NEW to RCDATA. The commands to change the globals of RCDATA are:

```
EX CMD RCDATA
EXTRACT NEW,TEMP
DEL RCDATA.TXT
RENAME NEW.TXT RCDATA.TXT
```

If you have a program which has many modules, typing these four commands for each module is burdensome. Also all of these modules must be re-compiled and re-assembled. To completely automate the whole process of changing globals, I have created a program called GEN which creates an EXEC file called NW GLOB. GEN prompts for a list of files. The four commands to change the globals plus a command to re-compile plus a command to re-assemble are written to NW GLOB as you enter your list of modules. Finally the one simple command :

```
EXEC NW_GLOB
```

creates new globals in all of your modules, in both the source code and the object code. Below is the source code for EX_CMD and GEN.

```
PROGRAM EX_CMD(INPUT,OUTPUT);

(* DESCRIPTION: This program creates the directive file
used by the extract command when changing globals. The
input file name is entered in the command line. EX CMD
searches the input file for the line number where the
globals start and end. The start and end of the global
block is identified by the marker. It uses these line
numbers to create a directive file used by EXTRACT.
*)

VAR
  IN_FILE : STRING[8];
  OUT_FILE : STRING[8];
  FILE_DEVICE : TEXT;
  LINE_NO : INTEGER;
  LINE : STRING;
  START, FIN : INTEGER;
  MARKER : STRING [3];
  MATCH : INTEGER;

PROCEDURE STOP;

BEGIN
  WRITELN ('UNABLE TO FIND GLOBAL MARKER');
  CLOSE (FILE_DEVICE);
  DEL (FILE_DEVICE, 'TEMP')
END;

PROCEDURE FIND;
BEGIN
REPEAT
  READLN (FILE_DEVICE, LINE);
  LINE_NO := SUCC (LINE_NO);
  MATCH := INDEX (LINE, MARKER)
UNTIL (MATCH <> 0) OR (EOF (FILE_DEVICE));
END;

BEGIN

LINE_NO := 0;
IN_FILE := CLINE;
OPEN (FILE_DEVICE, IN_FILE, INPUT);
MARKER := '(*>';

FIND;
START := LINE_NO;
IF EOF (FILE_DEVICE) THEN STOP
ELSE
  BEGIN
  MARKER := '<*)';
  FIND;
  FIN := LINE_NO;
  IF EOF (FILE_DEVICE) THEN STOP
  ELSE
    BEGIN
    CLOSE (FILE_DEVICE);
    OUT_FILE := 'TEMP';
    CREATE (FILE_DEVICE, OUT_FILE, OUTPUT);
    WRITELN (FILE_DEVICE, ')', IN_FILE, ' -', START-1 :0);
    WRITELN (FILE_DEVICE, ')TEMPLATE');
    WRITELN (FILE_DEVICE, ')', IN_FILE, ' ', FIN+1 :0, '-');
    CLOSE (FILE_DEVICE)
    END
  END

END.
```

```
PROGRAM GEN (INPUT, OUTPUT);
(*
DESCRIPTION : This program prompts for a list a files.
For each file it
            creates 6 commands:
               EX CMD file name
               EXTRACT NEW,TEMP
               DEL file name
               RENAME NEW.TXT, file name
               PC <file name >file name 0
               RA <file name >file name 0
             GEN outputs these commands to a file called
NW_GLOB which
             is used as an EXEC file to change globals.
*)

VAR
  OUTFILE : TEXT;
  FILENAME, NAME : STRING;

BEGIN

FILENAME := 'NW GLOB';
REWRITE (OUTFILE, FILENAME);
WRITELN ('TYPE "<QUIT" TO EXIT');
WRITE ('FILENAME: ');
READLN (NAME);
WHILE NAME <> '<QUIT' DO
  BEGIN
  WRITELN (OUTFILE, 'EX CMD ', NAME);
  WRITELN (OUTFILE, 'EXTRACT NEW, TEMP');
  WRITELN (OUTFILE, 'DEL ', NAME, '.TXT');
  WRITELN (OUTFILE, 'RENAME NEW.TXT ', NAME, '.TXT');
  WRITELN (OUTFILE,'ECHO COMPILING ',NAME);
  WRITELN (OUTFILE,'PC <',NAME,' >',NAME,' 0');
  WRITELN (OUTFILE,'ECHO ASSEMBLING ',NAME);
  WRITELN (OUTFILE,'RA <',NAME,'.CO >',NAME,' 0');
  WRITE ('FILENAME: ');
  READLN (NAME)
  END;
CLOSE (OUTFILE)
END.
```

BASIC'S "USER"

WOODY BAKER
RT 11 BOX 4780
LUFKIN, TX 75901
(713)-639-3417

USING USER

This article was first written over 2 years ago, just after I had purchased my TRS-80C compute . At that time, Radio Shack had not released their updated manual, and there was no information

available on how to use the USR function. The article was written and presented to our local computer club. Unfortunately, it went way over the heads of most of the people present, but I trust that our readership will be able to understand it.

We are going to take a look at the USR command and what it can accomplish for you, the programmer.

First things first, so let us look at the syntax for the USR command. The syntax is VARIABLE=USR(VARIABLE2 or CONSTANT). The value within the () is passed to the floating point accumulator for use by the machine code routine. The result that the user's machine language routine is returned to Basic through the same accumulator, and assigned to the variable located on the left side of the "=" sign. The variable on the left, is optional, in that the USR command can be used within an expression, and the result used by the expression, and then discarded. For this discussion we will assume that the variable is required.

The USR command leaves the value that is passed to it, in the floating point accumulator, located at $4F-$54. This is the primary accumulator for floating point operations, and is utilized in much the same way as the A accumulator with in the 6809 chip. If we desire to operate on the floating point value directly, we don't need to convert it to integer, but if we need the value within the machine language routine, then we must convert the value to a two byte integer. An example of operating on the value directly, is adding a number to the exponent byte to multiply the value by some power of 10. For example, if we add two to the floating point exponent, then we will have multiplied the number by 100. A little thought about this, will enable you to do very fast multiplies by powers of 10's, because the computer doesn't have to do a floating point multiply. If we desire to use the value in our machine code routine, we need to convert it from floating point representation to integer representation. This requires a lot of coding, but since this function is required by the basic interpreter, and is in the ROM, we don't have to worry about it at all. All we need to do, is to to a JSR to IFIX, which is located at location $BCC8, and the integer result returns in the X register and in locations $52 and $53, with $52 being the mist significant byte of the two byte number.

In order to have the value returned to Basic, it must be left as a floating point number in the floating point accumulator. If you wish to return an value from a machine code routine that is a two byte integer, you will have to convert it to floating point before you return to Basic. If you are just operating on the number within the floating point accumulator, you need only do a RTS as the number already is in floating point format. To convert a number from integer into floating point, we can again take advantage of subroutines living in the ROMS. There are two primary routines here, one which floates the D accumulator (the A and B accumulators combined) and one that floats only the B accumulator. The single byte float routine called FLOATB, is located at $B4F3. It simply clears the A accumulator and then falls into FLOATD, located at $B4F4, which floats the A and B accumulators.

Listing 1. shows one usage of the USR function, in this case an exclusive OR of the high byte and the low byte of a two byte integer number.

Listing 1. also has an example of a rather interesting technique for embedding machine code into a basic program. I stumbled on to this technique back in 1979, when working on a light pen for ESMARK INC. Essentially, it fools basic into thinking that a machine code program is a REM statement. It works as follows. Line 10 of the program consists of a REM statement, followed by 12 *'s. The program pokes the machine code into the place of the *'s. There must not be a space between the REM and the first *.

The machine code, is 12 bytes long, so there must be 12 *'s there. This REMARK statement in this case, must be the first line of the program, unless you change the offsets located in lines 15 and 16. The ofsets allow the program to determine where it is, regardless of where the start of basic is.

This technique of embedding machine code in REM statements, is very handy, because the code will load and save right along with the basic program. Due to the way that Microsoft stores its lines of basic code in memory by using pointers to the start of the next line, you can embed as much machine code as you wish in a REM line, if you change the pointers.

In practice, since the color computer allows you to type in up to 255 characters in a line, you can set up a 254 byte line with 1 guard byte left over, and all this without having to fix the pointers.

Line 15 of the program makes B point to the start of the basic workspace, adds an ofset of 5 bytes to it, and makes C point to the start of the *'s. Lines 17 and 18 read the machine code out of the data statements and poke the machine code into the line of *'s. It terminates when it encounters a 0 in the code. Using this technique, you cannot have a 0 byte in the machine code, as basic will find it and think that this is the end of the basic line. It will then set it's pointers up, and wipe out the next 4 bytes of code. Line 30 is the decimal equivalent of the machine code shown in listing 2. Line 35 calculates the two byte value to poke into the USR locations that tell the USR function where to go. In extended Color Basic, you would have to replace line 35 with the following line:

```
35 DEFUSR(0)=C:REM SET USR(0)=>MACHINE CODE.
```

Line 36 pokes the user address into the user vector for the user call. Line 40 sets x to 16706, which will convert to the two byte integer $4142, or 'AB'. This allows you to pass two one byte values to the USR routine. To do so, you must first multiply the first value by 256 and then add it to the second value. Upon return from USR(X), Y will hold the value 3, which is the exclusive or of $41 and $42. If you are using Extended Color Basic, you will have to replace USR(X) with USR0(x).

A word about the program. In cryptography, it is often desirable to exclusive or two characters together, one character being the key and the other character being the one to encode with the key. The result can be easily converted back to the original value, by simply exclusive oring the encoded byte with the key byte.

Someone may be asking, well how do I use this program to encode text data? Simply by taking a character from the key string, and exclusively oring it with the corresponding character from the string to be encoded. To do this, you must first isolate the two characters, convert them to their binary equivalent, multiply one by 256 and add it to the other, and then call USR(X), with the result in X. This task is then repeated until the two strings are exhausted of all characters.

Listing 4. provides a much simpler way of doing this. The instructions for setting it up are the same as for listing 1., except that it needs 52 *'s, as the machine code is longer. This routine is very different in function however. It takes the names of 2 strings the same length, passed to it in the above manner, and replaces the first string with

the exclusive or of the two strings. It actually looks the strings up in memory, and works directly on the strings. If the strings are of different length, the function call error routine is called. In order for this to work, the ascii equivalent of name of the string to BE encoded must be multiplied by 256 and the ascii equivalent of the KEY string name, must be added to it. The strings MUST be single character names, such as A$, B$ etc. but not A1$.

Listing 5. is the machine language routine that is used in listing 4. After running the program the first time, you may delete lines 28 to the end of the program, and save the resulting routine to tape for further use. Lines 30-110 are a demonstration of the way that the program is used. Beware of using a constant string for the destination string, as they are stored in the program, rather than in string memory and you could be in serious trouble if it got out of hand. If it produced a 0 anywhere, you could be in serious trouble. If you want to make sure that a string is held off in memory somewhere, you can either assign it to another string, or append another letter to it. This is why the string in listing 4., is 1 character shorter than the key string, and has a "+" appended to it. This forces the thing off into RAM. Line 60 of the program assigns the string names to the appropriate two byte value for the call.

The two assembly programs are commented enough to make them understandable, I hope. Listing 5. uses a new routine in the ROM. It is called FNDVAR, and it looks up a variable in the symbol table or variable table for basic. If the variable is not there, it is allocated blank storage in the table. The routine operates by finding the names of the variables to lookup in locations $37 and $38, and returns the address of the variable in locations $39 and $3A. The routine FNDVAR is located at $B38F.

I have shown you how to use the USR function on the color computer, and have given a practical example of it's usage. Happy USRing.

I am very deeply involved in dissecting the color computers basic, and I have a lot of information about it, and am always looking for more. If you need help with understanding it, feel free to contact me.
If enough interest is there, I might be persuaded to do some more articles on the ROMS. If you feel that you absolutely have to call me, please don't call between about 5 p.m. Friday (Texas time) through 9 p.m. Saturday, as I will not be available for comment or discussion during that time.

LISTING 1.

```
10 REM**************
15 B=PEEK(25)*256+PEEK(261):C=B+5:REM BASE OF BASIC
16 B=B+5:REM 5 BYTES TO FIRST *
17 READ A:IF A=0 GOTO 35:REM PLACE MACHINE CODE.
18 POKE B,A:B=B+1:GOTO17:REM NEXT LOCATION
30 DATA 189,180,200,79,214,82,216,83,189,180,244,57,0
35 C1=INT(C/256):C2=C-(C1*256):REM CALCULATE USER ADDRESS
36 POKE 275,C1:POKE276,C2
40 X=16706:Y=USR(X):PRINTY
```

LISTING 2.

```
        ORG     $0000     DOES NOT MATTER, POSITION Independant
XOR     JSR     $BCCB     GET INTEGER=>$52,$53
        LDA B   $52       GET 1/2
        EOR B   $53       EXCLUSIVE OR BOTH HALVES
        CLR A
        JSR     $B4F4     FLOAT D
        RTS

* ALTERNATE CODING.  NOT USED
        ORG     $0000
XOR     JSR     $BCCB
        LDA B   $52
        EOR B   $53
        JMP     $B4F3     FLOAT B
```

LISTING 4.

```
10 REM**************************************
20 REM ABOVE=52 BYTES OF MACHINE CODE.
25 C=PEEK(25)*256+PEEK(26):C=C+5:REM START OF CODE
26  C1=PEEK(25):C2=PEEK(26):REM CALC ADDRESS
27  POKE 275,C1:POKE 276,C2:REM SET JUMP VECTOR
28 GOSUB 300:REM DELETE 28-350 AFTER RUNNING PROGRAM
30 L$="THIS IS A TES":REM SO WE CAN FORCE IT TO RAM
40 C$="A NEW KEY TO U":REM KEY
50 B$=L$+"T":REM FORCE TO RAM
55 PRINT B$:REM SHOW STRING FIRST
60 Y=ASC(B$)*B+1*256+ASC("C"):REM SET NAMES UP  B$=ENCODED
70 X=USR(Y):REM X MUST BE THERE AS A DUMMY ARGUMENT
80 PRINT B$:REM ENCODED STRING
90 X=USR(Y):REM  DECODE IT. AS B IS NOW ENCODED..USE AS NEW TARGET.
100 PRINT B$
110 END
120 DATA 189,188,200,220,82,52,6,198,128,221,95,189,179,143,158
129 DATA 57,53,6,52,16,215,53,189,179,143,53,16,222,57,166,132
130 DATA 161,196,38,14,174,2,238,66,230,252,132,231,128,74,38,247
140 DATA 57,126,183,6,0
300 READ A:IF A=0 THEN 350:REM POKE CODE IN
320 POKE C,A:C=C+1:GOTO 300
350 RETURN
```

LISTING 5.

```
                ORG     $0000   DOESN'T MATTER. PIC
B0 BCCB STROR   JSR     $BCCB   GET NAMES
DC 52           LDD     $52     PUT THEM INTO D
34 06           PSHS    D       SAVE THEM
86 80           LDA B   $80     TURN FIRST INTO STRING
DD 37           STD D   $37     SET UP NAME
BD B38F         JSR     $B38F   JSR FNDVAR  RETURN POINTER IN$39,3A
9E 39           LDX     $39     GET FIRST POINTER
35 06           PULS D          GET NAMES BACK
34 10           PSHS X          SAVE LOCATION OF FIRST
D7 37           STA B $37       SET SECOND UP AS STRING..THE $80 STILL THERE
BD B38F         JSR     $B38F   FIND VAR
35 10           PULS X          GET ADDRESS OF FIRST
DE 39           LDU     $39     GET ADDRESS OF SECOND
A6 84           LDA A 0,X       CHECK FOR LENGTHS MATCHING.
A1 C4           CMP A 0,U
26 0E           BNE     EXIT    IF NO MATCH
AE 02           LDX     2,X     PICK UP ACTUAL ADDRESS OF STRING.
EE 42           LDU     2,X
E6 C0 LOOP      LDA B 0,U+      GET KEY AND ADVANCE POINTER
E8 84           EOR B 0,X       EXCLUSIVE OR THEM
E7 80           STA B 0,X+      SAVE AND INC POINTER
4A              DEC A           DECREMENT THE LENGTH
26 F7           BNE     LOOP    UNTIL DONE.
39              RTS
7E B706         JMP     FNCERR  FUNCTION CALL ERROR..
```

FORMAT OF STRING STORAGE IN VAR TABLE.

```
STRING NAME---------->FIRST CHARACTER OF STRING NAME
STRING NAME2--------->SECOND CHARACTER OF STRING NAME.
                      IT HAS $80 ADDED TO IT.
LENGTH--------------->LENGTH OF THE STRING
MSB ADDRESS---------->ACTUAL ADDRESS OF STRING IN MEMORY MSB.
LSB ADDRESS---------->ACTUAL ADDRESS OF STRING IN MEMORY LSB.
```

FNDVAR leaves the variable address pointing to the first byte AFTER the string name, which in this case is the length of the string.

CHECKERS

I've enclosed the following source listings for possible inclusion in one of your issues of '68' Micro Journal.

This is a complete checkers game program written in Microware's Basic09 language. This program really works, and it is complete. I wrote my own checkers program, out of disgust, when I discovered that all the existing or published programs were incomplete, lacked most of the features needed in playing checkers, and usually did not work, or worked at best marginally.

I designed this program to be very modular, thus it allows for considerable room for modifications and improvements in the program. I kept the evaluation procedure separate, to allow for trying out newer and better algorithims. The CRT board display procedures can be easily modified for most terminals without interfering with the other procedures.

After trying several other checkers programs on the market, and finding out that they aren't any better than this one, I am quite pleased with mine. Especially since the other checkers games cost $29.95 and up (these are the ones on the cheap arcade games or such items for so called personal home computers).

Also note the instructions on how to properly overlay procedures so that memory usage is minimized. The Basic09 manual does not actually state how to do this properly. When a procedure is packed it gets special I code names assigned to each variable and procedure names. But if you save each procedure separately, then the actual I code names will not match and you'll get ERROR CODE 43 (unknown procedure) generated. When you try and run the program or procedure.
By using a string variable and appending the procedure name (minus string quotes to trim constant), one can do wild overlaying the procedures and build up large procedure libraries for various functions easily. At the time, I must have been really stupid, as it took me about two weeks or so to figure it out. It was real interesting.

Answer here is a complete checkers game program that really works.

I really appreciate your publishing the previous two program submissions. I had sent to you.

Thank you very much.

Earl W. Bollinger

Earl W. Bollinger
10818 North 91st. Ave.
Peoria, Arizona 85345

602-979-4451

```
PROCEDURE CHECKERS
 0000       (* PROCEDURE CHECKERS  VER. 1.0                              *)
 003C       (* THIS IS THE MAIN EXECUTIVE PROCEDURE USED WITH ALL THE     *)
 0078       (* OTHER SUB-PROCEDURES. THAT THIS PROGRAM CALLS AS           *)
 00B4       (* NEEDED. WHEN IN USE.                                       *)
 00F6       (* THIS IS A COMPLETE CHECKERS GAME. IT ALLOWS FOR KINGS,     *)
 012C       (* SINGLE, DOUBLE AND TRIPLE JUMPS. AND WHEN THE GAME         *)
 0168       (* IS ENDED, IT WILL PROMPT YOU FOR ANOTHER GAME.             *)
 01A4       (* THE EVAL PROCEDURE USES A SINGLE PLY METHOD FOR            *)
 01E0       (* EVALUATION OF THE MOVES. THE COMPUTER'S SIDE OF THE        *)
 021C       (* GAME TENDS TO BE OF A BASICALLY DEFENSIVE NATURE.          *)
 0258       (* WRITTEN BY E.W.BOLLINGER ON JUNE 27, 1982                  *)
 0294       (* 10810 NORTH 91ST. AVE., PEORIA, ARIZONA 85345              *)
 02D0       (* FOR BASIC09 AND OS-9 WITH AT LEAST 56K BYTES OF RAM.       *)
 030C       (* GRAPHICS ARE FOR A Z-19 OR H-19 TERMINAL.                  *)
 0348       (* THIS PROGRAM REQUIRES AT LEAST 17K BYTES OF USER RAM       *)
 0384       (* TO USE ON MACH1  0 WITH LESS THAN 17K BYTES YOU MUST       *)
 03C0       (* MODIFY THE PROGRAM LIKE SO:                                *)
 03FC       (* DIM COMM:STRING                                            *)
 0438       (* THEN WHERE EACH RUN PROCEDURE MAKE COMMAND IS:             *)
 0474       (* COMM:="PROCEDURE NAME"                                     *)
 04B0       (* RUN COMM( PARAMETER LIST )                                 *)
 04EC       (* KILL COMM                                                  *)
 0528       (* THE PROCEDURES ARE THEN STORED IN THE CURRENT COMMAND      *)
 0564       (* DIRECTORY AS PACKED I CODE MODULES.                        *)
 05A0       BASE 0
 05A2       DIM BOARD(12,12):STRING(8)
 0607       DIM CLRSQ,WHSQ,BLACK,WHITE,WHKING,BLKING,EMPTY:STRING[10]
 06D8       DIM CURSOR,M,A,B,C,COMMAND:STRING[5]
 07FB       DIM I,K,J,L,N,RAN:INTEGER
 0816       DIM FLAG,OOPS:INTEGER
 0821       (* MAIN EXECUTIVE SECTION *)
 0830       (* INITIALIZE *)
 0840 10    PRINT CHR$( 7); CHR$(59);   \(* CLEAR SCREEN AND CURSOR HOME *)
 087C       CURSOR:=CHR$(27)+CHR$( 3)
 08B8       BLACK:=CHR$(27)+"Pbb"+CHR$(27)+"Q"
 08FE       WHITE:=CHR$(27)+"p  "+CHR$(27)+"q"
 0944       WHKING:=CHR$(27)+"pHX"+CHR$(27)+"q"
 098A       BLKING:="HX"
 0903       EMPTY:="  "
 09DC       (* INITIAL ARRAY SETUP *)
 09F5       RUN SETARRAY(BOARD,EMPTY,WHITE,BLACK)
 0A3E       (* COMPUTER MAKES THE FIRST MOVE *)
 0A31       RUN FIRSTMOVE(BOARD,EMPTY,BLACK,RAN)
 0A7A       (* DRAW CHECKERBOARD *)
 0A61       RUN DRAWBOARD(CURSOR)
 0A68       (* PRINT THE CURRENT PIECE POSITIONS *)
 0A92 50    RUN POSITIONS(BOARD,CURSOR)
 0AA4       (* INPUT COMMAND *)
 0AB6 60    A:=""

 0AC2       PRINT CURSOR; "60"; CHR$(27); CHR$(79);
 0AD5       INPUT "ENTER YOUR MOVE( ",A
 0AEE       (* CHECK FOR A BAD INPUT *)
 0B00       OOPS:=0
 0B1B       L:=LEN(A)
 0B19       FOR N=1 TO L
 0B2A         C:=MID$(A,N,1)
 0B38         IF C("0" THEN
 0B45           IF C()"-" THEN
 0B52             OOPS:=1
 0B59           ENDIF
 0B30         ENDIF
 0B50         IF C)"9" THEN
 0B6A           IF C()"X" THEN
 0B77             OOPS:=1
 0B7E           ENDIF
 0B00         ENDIF
 0B92       NEXT N
 0BB0       IF OOPS=1 THEN
 0B99         PRINT CHR$(7)
 0B9F         GOTO 60
 0BA3       ENDIF
 0BA5       IF LEN(A)()8 THEN
 0BB2         (* CORRECT FOR SINGLE DIGIT ENTRIES *)
 0BD0         N:=1
 0BDF         C:=""
 0BE6         REPEAT
 0BE0           C:=C+MID$(A,N,1)
 0B9A           N:=N+1
 0B03         UNTIL MID$(A,N,1)="-" OR MID$(A,N,1)="X"
 0B25         IF LEN(C)=1 THEN
 0B32           C:="0"+C
 0B3E           B:=C+MID$(A,N,1)
 0B50           N:=N+1
 0B5B         ELSE
 0B5F           B:=LEFT$(A,3)
 0B6A           N:=N+1
 0B75         ENDIF
 0B77         C:=""
 0B7E         REPEAT
 0B00           C:=C+MID$(A,N,1)
 0B92           N:=N+1
 0B30         UNTIL MID$(A,N,1)="0"
 0BAE         IF LEN(C)=1 THEN
 0BB0           C:="0"+C
 0BC7           B:=B+C
 0BD3           A:=B
 0BDD         ELSE
 0B0F           B:=B+C
 0BE3           A:=B
 0BF3         ENDIF
 0BF5       ENDIF
 0BF7       (* TEST FOR A BAD INPUT *)
 0A11       IF MID$(A,3,1)()"-" THEN
 0A23         IF MID$(A,3,1)()"X" THEN
 0A35           OOPS:=1
 0A3C         ENDIF
 0A3C       ENDIF
 0A48       IF LEFT$(A,2)="32" THEN
 0A51         OOPS:=1
 0A58       ENDIF
 0A5A       IF RIGHT$(A,2)="32" THEN
 0A68         OOPS:=1
 0A72       ENDIF
 0A74       IF LEFT$(A,2)("81" THEN
 0A85         OOPS:=1
 0A8C       ENDIF
 0A8B       IF RIGHT$(A,2)("81" THEN
 0A9F         OOPS:=1
 0AAC       ENDIF
 0AAE       IF OOPS=1 THEN
 0AB4         PRINT CHR$(7)
 0AB8         GOTO 60
 0AB6       ENDIF
 0AC0       RUN CONVERT(A)
 0AC0       (* CHANGE BOARD TO REFLECT THE NEW INPUT *)
 0AF5       RUN CHANGEARRAY(BOARD,A,WHKING,WHITE,EMPTY,M)

 0B10       (* PRINT CURRENT POSITIONS *)
 0B35       RUN POSITIONS(BOARD,CURSOR)
 0B44       (* CHECK FOR MULTIPLE JUMPS *)
 0B62       IF M="X" THEN 60
 0B72       (* COMPUTER'S MOVE EVALUATION *)
 0B92       RUN EVAL(BOARD,FLAG)
 0BA1       (* IF END OF GAME CHECK *)
 0BB0       IF FLAG=50 THEN
 0BC         PRINT "THANK YOU FOR A VERY ENJOYABLE GAME."
 0BDF        PRINT
 0BF1        PRINT "DO YOU WANT TO PLAY ANOTHER?"
 0C11        PRINT
 0C13        INPUT "ENTER A Y/N: ",COMMAND
 0C28        IF COMMAND="Y" THEN
 0C35          GOTO 10
 0C39        ENDIF
 0C38        IF COMMAND="N" THEN
 0C48          GOTO 100
 0C4C        ENDIF
 0C4E       ENDIF
 0C50       GOTO 30
 0C54 100   END
PROCEDURE SETARRAY

 0000       (* THIS PROCEDURE IS CALLED BY THE CHECKERS EXECUTIVE *)
 0038       (* PROCEDURE FOR USE IN SETTING UP THE CHECKERS BOARD *)
 0078       (* WRITTEN BY E.W.BOLLINGER ON JUNE 27, 1982          *)
 00AB       BASE 0
 00A0       PARAM BOARD(12,12):STRING(8)
 00BF       PARAM EMPTY,WHITE,BLACK:STRING[10]
 00D3       DIM CURSOR:STRING[5]
 00DF       DIM RAN,I,J:INTEGER
 00EE       DIM ACNTR,BCNTR:INTEGER
 00F9       DIM ALINE,BLINE:STRING[85]
 0109       DIM CLRSQ,WHSQ:STRING[10]
 0119       (* SETUP THE BOARD *)
 012E       FOR I=2 TO 9
 0136         FOR J=2 TO 9
 014E           BOARD(I,J):=EMPTY
 0150         NEXT J
 0168       NEXT I
 0173       FOR I=2 TO 4 STEP 2
 0188         FOR J=2 TO 8 STEP 2
 0190           BOARD(I,J):=WHITE
 01AC         NEXT J
 01B7       NEXT I
 01C         FOR I=3 TO 3
 01D           FOR J=3 TO 9 STEP 2
 01E7            BOARD(I,J):=WHITE
 01F6         NEXT J
 0201       NEXT I
 020C       FOR I=7 TO 9 STEP 2
 0221         FOR J=3 TO 9 STEP 2
 0236           BOARD(I,J):=BLACK
 0265         NEXT J
 0250       NEXT I
 0259       FOR I=6 TO 8
 0268         FOR J=2 TO 8 STEP 2
 0280           BOARD(I,J):=BLACK
 02B7         NEXT J
 02BA       NEXT I
 02A5       FOR I=0 TO 11
 02B9         FOR J=0 TO 1
 02C5           BOARD(I,J):=""
 02D3         NEXT J
 02DE       NEXT I
 02E8       FOR I=0 TO 11
 02F9         FOR J=10 TO 11
 0309           BOARD(I,J):=""
 0317         NEXT J
 0322       NEXT I
 032B       FOR I=0 TO 1
 033D         FOR J=0 TO 11
 034D           BOARD(I,J):=""


 0358         NEXT J
 0366       NEXT I
 0371       FOR I=10 TO 11
 0381         FOR J=0 TO 11
 0391           BOARD(I,J):=""
 039F         NEXT J
 03AA       NEXT I
 03B5       END
PROCEDURE FIRSTMOVE
 0000       (* THIS PROCEDURE IS CALLED BY THE CHECKERS EXECUTIVE *)
 0038       (* PROCEDURE WHEN THE COMPUTER IS TO MAKE THE FIRST   *)
 0078       (* MOVE.  WRITTEN BY E.  BOLLINGER JUNE 27, 1982      *)
 00A8       BASE 0
 00AA       PARAM BOARD(12,12):STRING(8)
 00BF       PARAM EMPTY,BLACK:STRING[10]
 00CF       DIM RAN:INTEGER
 00D6 100   RAN:=RND(4)
 00E3       ON RAN GOTO 100,110,120,130,140
 00FE 110   BOARD(7,3):=EMPTY
 010E       BOARD(6,4):=BLACK
 0119       GOTO 200
 011F 120   BOARD(7,5):=EMPTY
 012F       BOARD(6,4):=BLACK
 013C       GOTO 200
 0148 130   BOARD(7,5):=EMPTY
 0158       BOARD(6,6):=BLACK
 0150       GOTO 200
 0161 140   BOARD(7,7):=EMPTY
 0171       BOARD(6,6):=BLACK
 017E 200   END
PROCEDURE DRAWBOARD
 0000       (* THIS PROCEDURE IS CALLED BY THE CHECKERS EXECUTIVE *)
 0038       (* PROCEDURE WHEN NEEDED.                             *)
 0078       (* WRITTEN BY E.W.BOLLINGER ON JUNE 27,1982           *)
 00AB       PARAM CURSOR:STRING[5]
 00B4       DIM ACNTR,BCNTR:INTEGER
 00BF       DIM ALINE,BLINE:STRING[85]
 00CF       DIM CLRSQ,WHSQ:STRING[10]
 00DF       CLRSQ:="      "
 00EC       WHSQ:=CHR$(27)+CHR$(78)+";;;;;;"+CHR$(27)+CHR$(71)
 0109       ALINE:=CLRSQ+WHSQ+CLRSQ+WHSQ+CLRSQ+WHSQ+CLRSQ+WHSQ
 012D       BLINE:=WHSQ+CLRSQ+WHSQ+CLRSQ+WHSQ+CLRSQ+WHSQ+CLRSQ
 0151       (* DRAW THE CHECKERBOARD *)
 016C       ACNTR:=1
 0173       WHILE ACNTR(5 DO
 017F         BCNTR:=1
 0186         WHILE BCNTR(4 DO
 0192           PRINT BLINE
 0197           BCNTR:=BCNTR+1
 01A2         ENDWHILE
```

```
01A6        SCNTR:=1
01AD        WHILE SCNTR<4 DO
0189          IF RCNTR=4 THEN
01C5            IF SCNTR=3 THEN
01D1              PRINT ALINE;
01D7              GOTO 180
01DB            ENDIF
01DD          ENDIF
01DF          PRINT ALINE
01E4 180      SCNTR:=SCNTR+1
01F2        ENDWHILE
01F6        RCNTR:=RCNTR+1
0201      ENDWHILE
0205      (* PRINT THE SQUARE NUMBERS ON THE BOARD *)
0230      PRINT CURSOR$; " ";28"; CURSOR$; " ";30"; CURSOR$; " ";31"; CURSOR
          $; " ";32";
025E      PRINT CURSOR$; "";25"; CURSOR$; "";26"; CURSOR$; "";27"; CURSOR
          $; "";28";
029C      PRINT CURSOR$; "";21"; CURSOR$; "";22"; CURSOR$; "";23"; CURSOR
          $; "";24";
0290      PRINT CURSOR$; "";17"; CURSOR$; "";18"; CURSOR$; "";19"; CURSOR
          $; "";20";
02E8      PRINT CURSOR$; "";13"; CURSOR$; "";14"; CURSOR$; "";15"; CURSOR
          $; "";16";
0316      PRINT CURSOR$; "";9 "; CURSOR$; "";10"; CURSOR$; "";11"; CURSOR
          $; "";12";
0344      PRINT CURSOR$; "";5 "; CURSOR$; "";6 "; CURSOR$; "";7 "; CURSOR
          $; "";8 ";
0372      PRINT CURSOR$; "";1 "; CURSOR$; "";2 "; CURSOR$; "";3 "; CURSOR
          $; "";4 ";
03A0      (* PRINT THE COMMAND INPUT SECTION OF THE SCREEN *)
03D0      PRINT CURSOR$; "";"; "CHECKERS VER. 1.0";
03F3      PRINT CURSOR$; "";"; "TO MAKE A MOVE:";
0410      PRINT CURSOR$; "";"; "Enter a From SQ# - To SQ#";
0437      PRINT CURSOR$; "";"; "FOR MULTIPLE JUMPS:";
0458      PRINT CURSOR$; "";"; "Enter a From SQ# X To SQ#";
047F      END
PROCEDURE POSITIONS
0000      (* THIS PROCEDURE UPDATES THE CURRENT PIECE POSITIONS *)
0038      (* ON THE CHECKERBOARD. IT IS CALLED BY THE CHECKERS  *)
0070      (* EXECUTIVE PROCEDURE WHEN REQUIRED.                 *)
00A8      (* WRITTEN BY E.W.BOLLINGER ON JUNE 27, 1982          *)
00E0      BASE 0
00E2      PARAM BOARD(12,12):STRING(8)
00F7      PARAM CURSOR$:STRING(6)
0103      DIM ROW,COLUMN,X,Y,RCNTR,SCNTR:INTEGER
011E      (* EVEN SQUARES *)
0130      ROW:=34
0137      X:=2
013E      WHILE ROW<35 DO
014A        COLUMN:=34
0151        Y:=2
0158        WHILE COLUMN<72 DO
0164          PRINT CURSOR$; CHR$(ROW); CHR$(COLUMN);
0174          PRINT BOARD(X,Y);
0180          COLUMN:=COLUMN+12
0189          Y:=Y+2
0196        ENDWHILE
019A        ROW:=ROW-6
01A5        X:=X+2
01B0      ENDWHILE
01B4      (* ADD SQUARES *)
01C5      ROW:=51
01CC      X:=3
01D3      WHILE ROW<52 DO
01DF        COLUMN:=48
01E6        Y:=3
01ED        WHILE COLUMN<78 DO
01F9          PRINT CURSOR$; CHR$(ROW); CHR$(COLUMN);
0209          PRINT BOARD(X,Y);
0215          COLUMN:=COLUMN+12
0228          Y:=Y+2
0229        ENDWHILE
022F        ROW:=ROW-6
023A        X:=X+2
0245      ENDWHILE
0249      END
PROCEDURE CONVERT
0000      (* THIS PROCEDURE SIMPLY TAKES THE STRING 'A' AND CONVERTS *)
0038      (* THE NUMBERS TO THE PROPER ARRAY VALUES B FOR USE BY THE  *)
007A      (* COMPUTER IN PLAYING THE CHECKERS GAME.                   *)
0087      PARAM A:STRING(6)
00C3      DIM C:STRING(2)
00CF      C:=LEFT$(A,2)
00DA      GOSUB 10
00DE      A:=C+RIGHT$(A,3)
00ED      C:=RIGHT$(A,2)
00FB      GOSUB 10
0105      A:=LEFT$(A,3)+C
0100      END
0100 10   IF C="A1" THEN
011E        C:="22"
0127        GOTO 20
0128      ENDIF
0128      IF C="B2" THEN
0138        C:="24"
0144        GOTO 20
0148      ENDIF
014A      IF C="B3" THEN
0158        C:="26"
0161        GOTO 20
0165      ENDIF
0167      IF C="B4" THEN
0175        C:="28"
017E        GOTO 20
0182      ENDIF
0184      IF C="B5" THEN
0192        C:="33"
            GOTO 20
019F      ENDIF
01A1      IF C="B6" THEN
01AF        C:="35"
01B8        GOTO 20
01BC      ENDIF
01BE      IF C="B7" THEN
01CC        C:="37"
01D5        GOTO 20
01D9      ENDIF
01DB      IF C="B8" THEN
01E9        C:="39"
01F2        GOTO 20
01F6      ENDIF
01F8      IF C="B9" THEN
0206        C:="42"
020F        GOTO 20
0213      ENDIF
0215      IF C="10" THEN
0223        C:="44"
022C        GOTO 20
0230      ENDIF
0232      IF C="11" THEN
0240        C:="46"
0249        GOTO 20
0240      ENDIF
024F      IF C="12" THEN
0250        C:="48"
0266        GOTO 20
026A      ENDIF
026C      IF C="13" THEN
027A        C:="53"
0283        GOTO 20
0287      ENDIF
0289      IF C="14" THEN
0297        C:="55"
02A0        GOTO 20
02A4      ENDIF
02A6      IF C="15" THEN
02B4        C:="57"
02B0        GOTO 20
02C1      ENDIF
02C3      IF C="16" THEN
02D1        C:="59"
02DA        GOTO 20
02DE      ENDIF
02E0      IF C="17" THEN
02EE        C:="62"
02F7        GOTO 20
02FB      ENDIF
02FD      IF C="18" THEN
0309        C:="64"
0314        GOTO 20
0318      ENDIF
031A      IF C="19" THEN
0328        C:="66"
0331        GOTO 20
0335      ENDIF
0337      IF C="20" THEN
0345        C:="68"
034E        GOTO 20
0352      ENDIF
0354      IF C="21" THEN
0362        C:="73"
036B        GOTO 20
036F      ENDIF
0371      IF C="22" THEN
037         C:="75"
0388        GOTO 20
038C      ENDIF
038E      IF C="23" THEN
039C        C:="77"
03A5        GOTO 20
03A9      ENDIF
03AB      IF C="24" THEN
03B9        C:="79"

03C8        GOTO 20
03CA      ENDIF
03CE      IF C="31" THEN
03DC        C:="97"
03E5        GOTO 20
03E9      ENDIF
03EB      IF C="32" THEN
03A1        C:="99"
04AA        GOTO 20
04AE      ENDIF
04B0 20   RETURN
PROCEDURE CHANGEARRAY
0038      (* THIS PROCEDURE CHANGES THE BOARD ARRAY TO MATCH THE *)
0039      (* CURRENT INPUTTED PIECE MOVE(S).                     *)
0072      (* WRITTEN BY E.W.BOLLINGER. JUNE 29,1982              *)
00A0      BASE 0
00A0      PARAM BOARD(12,12):STRING(8)
00C2      PARAM A:STRING(6)
00CE      PARAM WHKING,WHITE,EMPTY:STRING(18)
00E2      PARAM M:STRING(6)
00EE      DIM I,J,K,L:INTEGER
0101      I:=VAL(LEFT$(A,1))
010E      J:=VAL(MID$(A,2,1))
011D      K:=VAL(MID$(A,4,1))
012C      L:=VAL(RIGHT$(A,1))
0139      M:=MID$(A,3,1)
0146      IF K=I-2 THEN
0156        IF L=J-2 THEN
0166          BOARD(K,L):=WHKING
0175          BOARD(I,J):=EMPTY
0184          BOARD(I-1,J-1):=EMPTY
0199          GOTO 200
019D        ENDIF
019F      ENDIF
01A1      IF K=I-2 THEN
01B1        IF L=J+2 THEN
01C1          BOARD(K,L):=WHKING
01D0          BOARD(I,J):=EMPTY
01DF          BOARD(I-1,J+1):=EMPTY
01F4          GOTO 200
01F8        ENDIF
01FA      ENDIF
01FC      IF K=9 THEN
0208        BOARD(K,L):=WHKING
0217        GOTO 100
0218      ENDIF
021D      IF BOARD(I,J)=WHKING THEN
0230        BOARD(K,L):=WHKING
023F        GOTO 100
0243      ENDIF

0249      BOARD(K,L):=WHITE
0254 100  BOARD(I,J):=EMPTY
0266      IF K=I+2 THEN
0276        IF L=J+2 THEN
0286          BOARD(K-1,L-1):=EMPTY
0290          GOTO 200
0297        ENDIF
02A1      ENDIF
02A3      IF K=I+2 THEN
02B3        IF L=J-2 THEN
02C3          BOARD(K-1,L+1):=EMPTY
02D8        ENDIF
02DA      ENDIF
02DC 200  END
PROCEDURE EVAL
0000      (* THIS PROCEDURE PERFORMS THE EVALUATION OF THE MOVES *)
0039      (* FOR THE COMPUTER'S PART OF THE GAME.                *)
0072      (* WRITTEN BY E.W.BOLLINGER ON JUNE 29, 1982.          *)
00AB      BASE 0
00AD      PARAM BOARD(12,12):STRING(8)
00C2      PARAM FLAG:INTEGER
00C9      DIM WHITE,BLACK,EMPTY,WHKING,BLKING:STRING(6)
00E5      DIM I,J,K,L:INTEGER
00F8      BLACK:=CHR$(27)+"F  "+CHR$(27)+"D"
010E      WHITE:=CHR$(27)+"F  "+CHR$(27)+"A"
012A      WHKING:=CHR$(27)+"FKK"+CHR$(27)+"A"
013A      BLKING:="KK"
0143      EMPTY:="  "
014C      (* BLACK CHECKING FOR JUMPS *)
016A 100  FLAG:=0
0174      I:=2
017B      LOOP
017D        J:=2
0184        LOOP
0186          IF BOARD(I,J)=BLACK THEN 150
019C          IF BOARD(I,J)<>BLKING THEN 190
01B2          GOSUB 3000
01B6        EXITIF FLAG>0 THEN
01C2          FLAG:=10
01C9        ENDEXIT
01C9          GOTO 190
01D1 150    GOSUB 2000
01D8        EXITIF FLAG>0 THEN
0164        ENDEXIT
0188 190      J:=J+1
01F6        EXITIF J>9 THEN
0202        ENDEXIT
0206        ENDLOOP
020A      EXITIF FLAG>0 THEN
0215      ENDEXIT
```

```
021A         I:=I+1
0225       EXITIF I>9 THEN
0231       ENDEXIT
0235       ENDLOOP
0239       (* MULTIPLE JUMPS *)
024D       (* KING JUMPS *)
025D       IF FLAG=10 THEN
0269         I:=K
0271         J:=L
0279         GOSUB 6000
027D       ENDIF
027F       IF FLAG=11 THEN
0289         I:=K
0291         J:=L
02 3         GOSUB 3000
0298       ENDIF
029F       IF FLAG=12 THEN
02A1         I:=K
02A9         J:=L
02B0         GOSUB 3000
02C1       ENDIF
02C3       (* REGULAR JUMPS *)
02D6       IF FLAG=1 THEN
02E2         I:=K
02EA         J:=L
02F2         GOSUB 6000
02F6       ENDIF
02FB       IF FLAG=2 THEN
0304         I:=K
030C         J:=L
0314         GOSUB 3000
0318       ENDIF
031A       IF FLAG=3 THEN
0326         I:=K
032E         J:=L
0336         GOSUB 2000
033A       ENDIF
033C       IF FLAG>0 THEN 500
034B       (* GUARDING AGAINST ANY JUMPS *)
035B       FLAG:=0
0372       I:=2
0379       LOOP
0378         J:=2
0302         LOOP
038A           IF BOARD(I,J)<>BLACK THEN 210
039A           GOTO 230
039E 210       IF BOARD(I-1,J-1)=WHITE THEN
03BA             IF      D(I+(-1,J+(-1)=EMPTY THEN
03D3               GOTO 230
0307             ENDIF
0309           ENDIF
0309           IF BOARD(I-1,J+1)=WHITE THEN
03FA             IF BOARD(I+1,J-1)<>EMPTY THEN
0480               GOTO 230
0411             ENDIF
0413           ENDIF
0415           GOTO 230
0419 230       IF BOARD(I+2,J)=BLACK THEN
0432             BOARD(I+2,J):=EMPTY
0444             BOARD(I+1,J+1):=BLACK
0459             FLAG:=FLAG+1
0464             GOTO 290
0468           ENDIF
046A           IF BOARD(I+2,J+2)=BLACK THEN
0483             BOARD(I+2,J+2):=EMPTY
0498             BOARD(I+1,J+1):=BLACK
04AD             FLAG:=FLAG+1
04B8             GOTO 290
04BC           ENDIF
04BE 230       IF BOARD(I+2,J-2)=BLACK THEN
04DA             BOARD(I+2,J-2):=EMPTY
04EF             BOARD(I+1,J-1):=BLACK
0504             FLAG:=FLAG+1
050F             GOTO 290
0513           ENDIF
0515           IF BOARD(I+2,J)=BLACK THEN
0528             BOARD(I+2,J):=EMPTY
053B             BOARD(I+1,J-1):=BLACK
0552             FLAG:=FLAG+1
055D             GOTO 290
0561           ENDIF
0563 230     EXITIF FLAG>0 THEN
0572       ENDEXIT
0576         J:=J+1
0581       EXITIF J>9 THEN
058D       ENDEXIT
0591       ENDLOOP
0595     EXITIF FLAG>0 THEN
09A1     ENDEXIT
05A5       I:=I+1
05B0     EXITIF I>9 THEN
05BC     ENDEXIT
05C0     ENDLOOP
05C4     IF FLAG>0 THEN 900
05D3     (* BLACK LOOKING FOR A SAFE MOVE *)
05F6     FLAG:=0
05FD     I:=2
0604     LOOP
0606       J:=2
0600       LOOP
0607         IF BOARD(I,J)=BLACK THEN 320
0625         GOTO 390
0629 320     IF BOARD(I-2,J)=WHITE THEN
0642           IF BOARD(I,J-2)<>EMPTY THEN
0658             GOTO 350
065C           ENDIF
065E         ENDIF
0660         IF BOARD(I-2,J-2)=WHITE THEN
0679           GOTO 390
067D         ENDIF
067F         IF BOARD(I-1,J-1)<>EMPTY THEN
0698           BOARD(I,J):=EMPTY
06A7           BOARD(I-1,J-1):=BLACK
06BC           FLAG:=FLAG+1
06C7           GOTO 390
06C9         ENDIF
06CB         IF BOARD(I-2,J)=WHITE THEN
06CD 350       IF BOARD(I,J+2)<>EMPTY THEN
06E6             GOTO 390
06FC           ENDIF
0700         ENDIF
0702         IF BOARD(I-2,J+2)=WHITE THEN 390
0720         IF BOARD(I-1,J+1)<>EMPTY THEN
0739           BOARD(I,J):=EMPTY
0748           BOARD(I-1,J+1):=BLACK
075D           FLAG:=FLAG+1
0768           GOTO 390
076C         ENDIF
076E 390   EXITIF FLAG>0 THEN
0778       ENDEXIT
0781         J:=J+1
078C       EXITIF J>9 THEN
0798       ENDEXIT
079C       ENDLOOP
07A8     EXITIF FLAG>0 THEN
07AC     ENDEXIT
07B8       I:=I+1
07B8     EXITIF I>9 THEN
07C7     ENDEXIT
07CB     ENDLOOP
07CF     IF FLAG>0 THEN 900
07DE     (* BLACK KING LOOKING FOR A SAFE MOVE *)
0803 488   FLAG:=0
080F       I:=2
0816       LOOP
0818         J:=2
081F         LOOP
0821           IF BOARD(I,J)=BLKING THEN 490
0837           IF BOARD(I-1,J+1)=EMPTY THEN
0858             IF BOARD(I-2,J)=WHITE THEN

0886                 IF BOARD(I-2,J+2)=WHITE THEN
0879                   GOTO 450
0883                 ENDIF
0885               ENDIF
0887             ENDIF
0889 410         IF BOARD(I-1,J-1)=EMPTY THEN
0898               IF BOARD(I-2,J)=WHITE THEN
08B8                 IF BOARD(I-2,J-2)=WHITE THEN
08D4                   GOTO 460
08D8                 ENDIF
08DA               ENDIF
08DC             ENDIF
08DE 420         IF BOARD(I+1,J+1)<>EMPTY THEN
08FA               IF BOARD(I+2,J)=WHITE THEN
0918                 IF BOARD(I+2,J+2)=WHITE THEN
0929                   GOTO 470
092D                 ENDIF
092F               ENDIF
0931             ENDIF
0933 430         IF BOARD(I+1,J-1)<>EMPTY THEN
094F               IF BOARD(I+2,J-2)=WHITE THEN
0968                 IF BOARD(I+2,J)=WHITE THEN
097E                   GOTO 480
0982                 ENDIF
0984               ENDIF
0986             ENDIF
0988             GOTO 490
098C 450         IF BOARD(I-2,J)=BLKING THEN
09A5               IF BOARD(I-2,J+2)=BLKING THEN
09BE                 IF BOARD(I,J+2)=BLKING THEN
09D4                   BOARD(I,J):=EMPTY
09E3                   BOARD(I-1,J+1):=BLKING
09F8                   FLAG:=FLAG+1
0A03                   GOTO 490
0A07                 ENDIF
0A09               ENDIF
0A0B             ENDIF
0A0B             GOTO 410
0A11 460         IF BOARD(I-2,J)=BLKING THEN
0A2A               IF BOARD(I-2,J-2)=BLKING THEN
0A43                 IF BOARD(I,J-2)=BLKING THEN
0A59                   BOARD(I,J):=EMPTY
0A68                   BOARD(I-1,J-1):=BLKING
0A7D                   FLAG:=1
0A84                   GOTO 490
0A88                 ENDIF
0A8A               ENDIF
0A8C             ENDIF
0A8C             GOTO 420
0A92 470         IF BOARD(I+2,J)=BLKING THEN

0AA9                 IF BOARD(I+2,J+2)=BLKING THEN
0AC4                   IF BOARD(I,J+2)=BLKING THEN
0ADA                     BOARD(I,J):=EMPTY
0AE9                     BOARD(I+1,J+1):=BLKING
0AFE                     FLAG:=FLAG+1
0B09                     GOTO 490
0B0A                   ENDIF
0B0C                 ENDIF
0B11               ENDIF
0B13               GOTO 430
0B17 480           IF BOARD(I+2,J-2)=BLKING THEN
0B33                 IF BOARD(I+2,J)=BLKING THEN
0B49                   IF BOARD(I,J-2)=BLKING THEN
0B5F                     BOARD(I,J):=EMPTY
0B6E                     BOARD(I+1,J-1):=BLKING
0B83                     FLAG:=FLAG+1
0B8E                     GOTO 490
0B92                   ENDIF
0B94                 ENDIF
0B96               ENDIF
0B98 490       EXITIF FLAG>0 THEN
0BA7           ENDEXIT
0BA8             J:=J+1
0BB6           EXITIF J>9 THEN
0BC2           ENDEXIT
0BC6           ENDLOOP
0BCA         EXITIF FLAG>0 THEN
0BD6       ENDEXIT
0BDA         I:=I+1
0BE3       EXITIF I>9 THEN
0BF1       ENDEXIT
0BF5       ENDLOOP
0BF9       IF FLAG>0 THEN 900
0C00       (* IF THERE IS NO SAFE MOVE *)
0C26       FLAG:=0
0C2D       I:=2
0C34       LOOP
0C35         J:=2
0C3D         LOOP
0C3F           IF BOARD(I,J)=BLACK THEN 520
0C55           GOTO 550
0C59 520       IF BOARD(I-1,J-1)<>EMPTY THEN
0C75             BOARD(I,J):=EMPTY
0C84             BOARD(I-1,J-1):=BLACK
0C98             FLAG:=1
0CA8             GOTO 550
0CAA           ENDIF
0CAC           IF BOARD(I-1,J+1)<>EMPTY THEN
0C9F             BOARD(I,J):=EMPTY
0CC2             BOARD(I-1,J+1):=BLACK
0CE3             FLAG:=FLAG+1
0CEE             GOTO 550
0CF2           ENDIF
0CF4 530     EXITIF FLAG>0 THEN
0D03         ENDEXIT
0D07           J:=J+1
0D12         EXITIF J>9 THEN
0D1E         ENDEXIT
0D22         ENDLOOP
0D26       EXITIF FLAG>0 THEN
0D32       ENDEXIT
0D36         I:=I+1
0D41       EXITIF I>9 THEN
0D4D       ENDEXIT
0D51       ENDLOOP
0D55       IF FLAG>0 THEN 900
0D64       FLAG:=0
0D6B       I:=2
0D72       LOOP
0D74         J:=2
0D7B         LOOP
0D7D           IF BOARD(I,J)=BLKING THEN 630
0D93           IF BOARD(I+1,J-1)=EMPTY THEN
0DAC             BOARD(I,J):=EMPTY
0DB8               D(I+1,J-1):=BLKING
0DC6             FLAG:=FLAG+1
0DC8             GOTO 680
0DDF           ENDIF
0DE1           IF BOARD(I+1,J+1)=EMPTY THEN
0DFA             BOARD(I,J):=EMPTY
0E09             BOARD(I+1,J+1):=BLKING
0E1E             FLAG:=FLAG+1
0E29             GOTO 680
0E2B           ENDIF
0E2D           IF BOARD(I-1,J-1)=EMPTY THEN
0E48             BOARD(I,J):=EMPTY
0E57             BOARD(I-1,J-1):=BLKING
0E6C             FLAG:=FLAG+1
0E77             GOTO 680
0E79           ENDIF
0E7B           IF BOARD(I-1,J-1)=EMPTY THEN
0E96             BOARD(I,J):=EMPTY
0EA5             BOARD(I-1,J+1):=BLKING
0EBA             FLAG:=FLAG+1
0EC5             GOTO 680
0EC9           ENDIF
0ECB 600     EXITIF FLAG>0 THEN
0EDA         ENDEXIT
0EDE           J:=J+1
0EE5         EXITIF J>9 THEN

0EF5         ENDEXIT
0EF7         ENDLOOP
0EFD       EXITIF FLAG>0 THEN
0F09       ENDEXIT
0F00         I:=I+1
0F18       EXITIF I>9 THEN
0F24       ENDEXIT
0F28       ENDLOOP
0F2C       IF FLAG>0 THEN 900
0F38       (* CHECK FOR MAKING BLACK KINGS *)
0F50 900   I:=2
0F57       J:=2
0F5E       WHILE J<=8 DO
0F78         IF BOARD(I,J)=BLACK THEN
0F80           BOARD(I,J):=BLKING
0F9C         ENDIF
0F9E         J:=J+2
0FA9       ENDWHILE
0FA9       (* CHECK FOR THE END OF GAME *)
0FCC       FLAG:=0
0FD3       I:=2
0FDA       LOOP
0FDC         J:=2
0FE3         LOOP
0FE3           IF BOARD(I,J)=WHITE THEN
0FF0             FLAG:=FLAG+1
1003             GOTO 950
1007           ENDIF
1009           IF BOARD(I,J)=BLKING THEN
101C             FLAG:=FLAG+1
1027             GOTO 950
1029           ENDIF
102D 930     EXITIF FLAG>0 THEN
103C         ENDEXIT
1040           J:=J+1
1048         EXITIF J>9 THEN
1057         ENDEXIT
1059         ENDLOOP
105F       EXITIF FLAG>0 THEN
106        ENDEXIT
1067           I:=I+1
107A       EXITIF I>9 THEN
1088       ENDEXIT
108A       ENDLOOP
108E       IF FLAG=0 THEN
1098         PRINT CHR$(27);CHR$(69);
10A4         PRINT "THE COMPUTER WINS!!!"
10BC         PRINT
10BE         FLAG:=50
10C3         GOTO 990
10C9       ENDIF
10CB       FLAG:=0
10D2       I:=2
10D9       LOOP
10DB         J:=2
10E2         LOOP
10EA           IF BOARD(I,J)=BLACK THEN
10F7             FLAG:=1
10FE             GOTO 990
1102           ENDIF
1104           IF BOARD(I,J)=BLKING THEN
1117             FLAG:=1
111E             GOTO 990
1122           ENDIF
1124 980     EXITIF FLAG>0 THEN
1133         ENDEXIT
1137           J:=J+1
1142         EXITIF J>9 THEN
114E         ENDEXIT
1152         ENDLOOP
1156       EXITIF FLAG>0 THEN
1162       ENDEXIT
1168         I:=I+1
1171       EXITIF I>9 THEN
117D       ENDEXIT
1181       ENDLOOP
```

```
1185        IF FLAG=0 THEN
1191          PRINT CHR$(27)+ CHR$(69)+
1198          PRINT "YOU WIN !!!!"
11A8          PRINT
11AD          FLAG:=58
11BA        ENDIF
11BE        END
11C0        (* REGULAR PIECE JUMPS SUBROUTINE *)
11C6 2000   IF BOARD(I-1,J+1)=WHITE THEN
11F8          IF BOARD(I-2,J+2)=EMPTY THEN
1211            BOARD(I-1,J+1):=EMPTY
1226            BOARD(I-2,J+2):=BLACK
1239            BOARD(I,J):=EMPTY
124A            FLAG:=FLAG+1
1255            K:=I-2
1260            L:=J+2
1268            GOTO 2050
126F          ENDIF
1271        ENDIF
1273        IF BOARD(I-1,J-1)=WHITE THEN
128C          IF BOARD(I-2,J-2)=EMPTY THEN
1295            BOARD(I-1,J-1):=EMPTY
12A8            BOARD(I-2,J-2):=BLACK
12BF            BOARD(I,J):=EMPTY

12DE            FLAG:=FLAG+1
12E9            K:=I-2
12FA            L:=J-2
12FF            GOTO 2050
1303          ENDIF
1305        ENDIF
1307 2050   RETURN
130C        (* BLACK KING JUMPS SUBROUTINE *)
132D 3000   IF BOARD(I-1,J+1)=WHITE THEN 3100
134C          IF BOARD(I-1,J+1)=WHKING THEN 3100
1368 3010   IF BOARD(I-1,J-1)=WHITE THEN 3120
1387          IF BOARD(I-1,J-1)=WHKING THEN 3120
13A3 3020   IF BOARD(I+1,J+1)=WHITE THEN 3130
13C2          IF BOARD(I+1,J+1)=WHKING THEN 3130
13DE 3030   IF BOARD(I+1,J-1)=WHITE THEN 3140
13F0          IF BOARD(I+1,J-1)=WHKING THEN 3140
1419        GOTO 3200
1418 3100   IF BOARD(I-2,J+2)=EMPTY THEN
1439          BOARD(I,J):=EMPTY
1448          BOARD(I-1,J+1):=EMPTY
1458          BOARD(I-2,J+2):=BLKING
1472          FLAG:=FLAG+1
147D          K:=I-2
1488          L:=J+2
1493          GOTO 3200
1497        ENDIF
1499        GOTO 3010
1498 3120   IF BOARD(I-2,J-2)=EMPTY THEN
14B9          BOARD(I,J):=EMPTY
14C8          BOARD(I-1,J-1):=EMPTY
14DD          BOARD(I-2,J-2):=BLKING
14F2          FLAG:=FLAG+1
14FD          K:=I-2
1508          L:=J-2
1513          GOTO 3200
1517        ENDIF
1519        GOTO 3020
1518 3130   IF BOARD(I+2,J+2)=EMPTY THEN
1539          BOARD(I,J):=EMPTY
1548          BOARD(I+1,J+1):=EMPTY
1558          BOARD(I+2,J+2):=BLKING
1572          FLAG:=FLAG+1
157D          K:=I+2
1588          L:=J+2
1593          GOTO 3200
1597        ENDIF
1599        GOTO 3030
1598 3140   IF BOARD(I+2,J-2)=EMPTY THEN
15B9          BOARD(I,J):=EMPTY
15C           BOARD(I+1,J-1):=EMPTY

15D8          BOARD(I+2,J-2):=BLKING
15F2          FLAG:=FLAG+1
15FD          K:=I+2
1608          L:=J-2
1613          GOTO 3200
1617        ENDIF
1619 3200   RETURN
```

BIT Bucket

UNIVERSAL DATA RESEARCH, INC. ANNOUNCES THE

TOTALLY FLEXIBLE REPORT WRITER II AND

NEW SOFTWARE PRICING

Universal Data Research, Inc. today announced a 20-25% average decrease in the prices of all applications software packages. UDRI recognizes the need for dependable, efficient business software at competitive prices, and is striving to make their software more easily available to the small and medium sized business by lowering prices and increasing the number of UDRI software dealers.

Universal Data Research, Inc. also announced today the Report Writer II, a totally flexible report generating program designed to enhance the capabilities of the UDRI Data Base Manager.

Some of the features available to the operator are:

- headers and footers for each page
- a wrap-up sheet for totalling columns and summaries
- 18 accumulators for any type of mathematical function
- generates reports involving linked files
- generates reports using a master file or its key file
- inclusion or exclusion of print lines if fields do not meet operator specified constrictions

This program greatly increases the flexibility of generating reports from the simple to the complex. Data can easily be placed virtually anywhere on an output line, allowing the user to run reports on preprinted forms without the need for costly software customization.

Report Writer II is available for $188.88 on the Flex* and UniFlex* operating systems. Expect to see this shortly on the Color Computer with Flex. Catalogue and complete price lists are available on request. Dealer inquiries are welcome.

For further information contact:

UNIVERSAL DATA RESEARCH, INC.
Cynthia Dylis Wilson
Director of Marketing
2457 Wehrle Drive
Buffalo, NY 14221
(716) 631-3011

* Flex and UniFlex are tradeworks of Technical Systems Consultants

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX 910-221-4055

Dan Williams
68 MICRO JOURNAL

02/17/83

Dear Dan:

Enclosed is your copy of OS-9 GMX II, Version 1.1. We are updating all current users of OS-9 GMX II (GIMIX version of OS-9 Level II) to this version at no charge. If any of your readers have not received their updates, plea s have them contact us directly.

GIMIX introduces the GMX III 6809 CPU board and OS-9 GMX III. The new CPU board is an advanced design, specifically intended for use with multi-user, multi-tasking operating systems. OS-9 GMX III is an enhanced OS-9 Level II that takes full advantage of the features of the new CPU board. The price for the combination of CPU board and software is $1650.00

Built on a multi-layer (6) circuit board and utilizing high-speed, high-density logic, the GMX III 6809 CPU board enhances the performance of the 2 MHz 6809 by providing such features as high-speed (1 byte/microsecond) DMA block transfers from memory to memory or between memory and I/O devices (such as the GIMIX Intelligent 3 Port Serial Interface) and advanced memory management with 2K segments and segment attributes. The board automatically arbitrates DMA contention between the on board DMA and external DMA devices such as disk controllers. The 2K memory segments allow more efficient memory usage. The segment attributes allow the trapping of out-of-range memory references (to protect one user's or task's memory from being accessed by another), write protection (to protect re-enable data and programs from modification which could affect the entire system), and a hardware single step function for software debugging (on a individual user basis without affecting other users or tasks).

The board prevents the execution of certain illegal instructions from crashing the system by monitoring interrupts to the 6809 and its response to them (these instructions cause the 6809 to lock up in a state in which it does not respond to any interrupts and must be reset). If the processor does not respond to an interrupt within a specific time (128 clock cycles) the board resets the 6809 (other devices on the bus are not reset) and asserts a special reset vector. The system can then close down the offending task and resume normal operation (other active tasks are not affected). This also limits the length of time that interrupts can remain masked by a user, preventing users from keeping the system from task switching and servicing other users.

To further protect the system from the users, the CPU board supports separate user and system "states" with automatic switching to the system state in response to interrupts and system (SWI) calls. Certain functions and memory areas can only be accessed in the system state, pr ven ing unauthorized accesses.

Also included on the new CPU are an improved full function time-of-day clock (MC146818) with year and automatic leap year/daylight savings time correction, and a 2K scratchpad RAM; both with battery backup standard. To provide precision timing functions, a 6840 PTM with a separate 500 KHz precision (.0025%) time base oscillator is included. The oscillator is easily user replaceable to provide other time base frequencies (750 KHz max.). The single EPROM socket will accept 2K, 4K or 8K EPROMs, with a maximum of 4K mapped into the system address space at any one time. Software switching is implemented by selecting the upper or lower half of an 8K EPROM under hardware or software control.

By taking advantage of the features of the GMX III CPU, OS-9 GMX III is faster, more memory efficient, and a more secure multi-user/multi-tasking operating system than OS-9 GMX II, from which it is derived. While retaining complete software compatibility. Throughput is enhanced by the ability to memory map and the automatic task switching, while the memory attributes and illegal instruction trapping protect the system and individual users from each other. Sharable system modules in RAM are write protected to prevent tampering. Memory mapping in 2K segments, and the ability to load modules in non-contiguous RAM provide more efficient memory utilization. Each task can be allocated a full 64K of RAM, with no operating system overhead in the tasks address space. Future plans for OS-9 GMX III include n optional hardware single stepping Debugger.

CHICAGO COMPUTER PRODUCTS
P.O. BOX 11943
CHICAGO, IL 60611-0943

'68' MICROJOURNAL
New Products Editor
P.O. Box 849
Hixson, TN 37343

26 February 1983

PRESS RELEASE INFORMATION

Chicago Computer Products announces two new products. One is a Four Digit Display Bareboard which has the following features:

- Plugs into one SS-30 slot
- Four digit seven segment LED display which is connected to the SS-30 card by means of a twelve conductor ribbon cable
- The segments of each display are separately controllable
- CMOS display controller chip
- Assembly manual and demonstration software examples provided

The other product is a controller barecard with the following features:

- All CMOS - low power consumption
- Uses the 146805E2P Motorola microprocessor for which there are cross assemblers available by other SS-30 bus suppliers
- 16 port lines, timer, clock, IRQ and RESET lines available at SS-30 card edge connector
- 3 - 2k by 8 bit RAM or EPROM. One must be EPROM.
- Assembly manual and diagnostic programs included.

The following was left on the 68 MICRO JOURNAL BBS. Thanks Kent.

Here is something for COCO users with 64K RAM and double sided disk drives: You can make Drive 2 the back side of Drive 0 and Drive 3 the back side of Drive 1.

With BASIC running in Ram type:

POKE &HD7AC,&H41 (for Drive 0)

POKE &HD7AD,&H42 (for Drive 1)

Now BASIC's Drive 2 is the back side of Drive 0 and Drive 3 is the back side of Drive 1. With this mod installed you can BACKUP your disks on the normally unused side. This works with the backup in the same drive or on the second drive. It would appear to really work great for someone with only one drive, if that drive is double sided. No switching disks. I haven't tried anything else using my new Drives 2 and 3 except COPY and DSKINI. Good Luck!!!

Kent.
KENT MEYERS. LE ROY,MN 55951
ATZ

ERNEST STEVE WATSON
11701 ST. CHARLES BLVD.
LITTLE ROCK, AR 72211

January 2, 1983

Dear Don:

I am enclosing a short LEX2/9(tm) utility called "HEXDEC.TXT" for publication. As its name indicates, it converts a four character hexidecimal number into its decimal equivalent. I find it very useful when programming in XBASIC and need to PEEK or POKE some decimal location for which I know the hexidecimal equivalent. It can be called while using TSC's XBASIC(tm) by using the "+" command.

I always look forward to receiving '68' Micro Journal. Thanks for an excellent publication.

Very truly yours,

Steve

FLEX and XBASIC are registered trademarks of Technical Systems Consultants,111 Providence Road, Chapel Hill, N. C. 27514

```
*A HEXIDECIMAL TO DECIMAL UTILITY
*FOR 6800 AND 6809 FLEX DOS

*FLEX 9.0 EQUATES

                CD1E  PSTRNG  EQU   $CD1E
                CD18  PUTCHR  EQU   $CD18
                CD15  GETCHR  EQU   $CD15
                CD03  WARMS   EQU   $CD03
                CD24  PCRLF   EQU   $CD24

C100                          ORG   $C100

C100                HONUM     RMB   2
C102                LONUM     RMB   2
C104                TEMP0     RMB   2
C106                TEMP1     RMB   2
C108                TEMP2     RMB   2
C10A                DVDN      RMB   2
C10C                DVSR      RMB   1
C10D                RMNDR     RMB   1
C10E                OUT       RMB   5
C113 04                       FC    4

C114 8E  C24B       START     LDX   #MSG1       PRINT HEADER
C117 BD  CD1E               JSR   PSTRNG
C11A BD  CD24               JSR   PCRLF
C11D 8E  C280               LDX   #MSG2
C120 BD  CD1E               JSR   PSTRNG       PRINT INSTRUCTIONS
C123 BD  C 24               JSR   PCRLF
C126 BD  CD15               JSR   GETCHR       GET HO ASCII NUMBER
C129 BD  C12E               JSR   TESTHEX
C12C 20  30                 BRA   NEXNUM

C12E 81  46       TESTHEX   CMPA  #$46         TEST FOR GREATER
C130 22  05                 BHI   ERR          THAN "F"
C132 81  30                 CMPA  #$30         TEST FOR LESS THAN "0"
C134 25  01                 BCS   ERR
C136 39                     RTS

C137 8E  C140       ERR       LDX   #ERRMSG
C13A BD  CD1E               JSR   PSTRNG
C13D 7E  CD03               JMP   WARMS
C140 45 4E 54 45  ERRMSG    FCC   "ENTER ONLY HEXIDECIMAL"
C144 52 20 4F 4E
C148 4C 59 20 48
C14C 45 58 49 44
C150 45 43 49 4D
C154 41 4C
C156 4E 55 4D 42            FCC   "NUMBE R"
C15A 45 52 53
C15D 04                     FCB   4

C15E B7  C100       NEXNUM    STAA  HONUM        STORE ASCII
C161 BD  CD15               JSR   GETCHR
C164 BD  C12E               JSR   TESTHEX
C167 B7  C101               STAA  HONUM+1
C16A BD  CD15               JSR   GETCHR
C16D BD  C12E               JSR   TESTHEX
C170 B7  C102               STAA  LONUM
C173 BD  CD15               JSR   GETCHR
C176 BD  C12E               JSR   TESTHEX
C179 B7  C103               STAA  LONUM+1
C17C 8E  C100               LDX   #HONUM
C17F BD  C18E               JSR   ASCBIN
C182 B7  C104               STAA  TEMP0
C185 BD  C18E               JSR   ASCBIN
C188 B7  C105               STAA  TEMP0+1
C18B 7E  C1ED               JMP   GORND

C18E 12             ASCBIN    NOP                CONVERT ASCII TO BINARY
C18F A6  84                 LDAA  X
C191 8D  1C                 BSR   CVERT
C193 1F  891D               TAB
C196 30  01                 INX
C198 A6  84                 LDAA  X
C19A 8D  08                 BSR   CVERT
C19C 58                     ASLB
C19D 58                     ASLB
C19E 58                     ASLB
C19F 58                     ASLB
C1A0 34  04 ABE0            ABA
C1A4 30  01                 INX
C1A6 39                     RTS

C1A7 84  7F         CVERT     ANDA  #$7F
C1A9 80  30                 SUBA  #$30
C1AB 81  0A                 CMPA  #10
C1AD 2B  02                 BMI   NOADD
C1AF 80  07                 SUBA  #7
C1B1 39             NOADD     RTS

C1B2 BF  C106       BNASC     STX   TEMP1
C1B5 30  01                 INX
C1B7 30  01                 INX
C1B9 30  01                 INX
C1BB 30  01                 INX
C1BD B7  C10A               STAA  DVDN
C1C0 F7  C10B               STAB  DVDN+1
C1C3 86  0A                 LDAA  #10
C1C5 B7  C10C               STAA  DVSR
C1C8 BF  C108       LOOP      STX   TEMP2
C1CB BD  C1FB               JSR   DIVBY
C1CE BE  C108               LDX   TEMP2
C1D1 B6  C10D               LDAA  RMNDR
C1D4 8B  30                 ADDA  #$30
C1D6 A7  84                 STAA  X
C1D8 30  1F                 DEX
```

```
C1DA BC  C106          CPX   TEMP1
C1DD 2C  E9            BGE   LOOP
C1DF BE  C106          LDX   TEMP1
C1E2 30  01            INX
C1E4 30  01            INX
C1E6 30  01            INX
C1E8 30  01            INX
C1EA 30  01            INX
C1EC 39                RTS

C1ED 8E  C10E  GORND   LDX   #OUT
C1F0 B6  C104          LDAA  TEMP0
C1F3 F6  C105          LDAB  TEMP0+1
C1F6 BD  C1B2          JSR   BNASC
C1F9 20  29            BRA   PRTMSG

C1FB 7F  C10D  DIVBY   CLR   RMNDR     DIVISION AND
C1FE 8E  0011          LDX   #17        REMAINDER
C201 20  08            BRA   RESTART
C203 B6  C10D  LOOP1   LDAA  RMNDR
C206 B0  C10C          SUBA  DVSR
C209 2A  04            BPL   NREST
C20B 1C  FE    RESTART CLC
C20D 20  05            BRA   MERGQ
C20F B7  C10D  NREST   STAA  RMNDR
C212 1A  01            SEC
C214 79  C10B  MERGQ   ROL   QVDN+1
C217 79  C10A          ROL   QVDN
C21A 30  1F            DEX
C21C 27  05            BEQ   RTN
C21E 79  C10D          ROL   RMNDR
C221 20  E0            BRA   LOOP1
C223 39        RTN     RTS

C224 8E  C296  PRTMS   LDX   #MSG1

C227 BD  CD1E          JSR   PSTRNG
C22A 86  20            LDAA  #$20
C22C BD  CD1E          JSR   PSTRNG

C22F 8E  C10E  P  OUT  LDX   #OUT
C232 BD  CD1E          JSR   PSTRNG
C235 8E  C2B1          LDX   #MSG4
C238 BD  CD1E          JSR   PSTRNG
C23B BD  CD24          JSR   PCRLF
C23E BD  CD15          JSR   GETCHR
C241 81  0D            CMPA  #$0D
C243 27  03            BEQ   DONE
C245 7E  C114          JMP   START
C248 7E  CD03  DON     JMP   WARMS

C24B 48 45 58 20  MSG1  FCC   "HEX TO DECIMAL CONVERSION - "
C24F 54 4F 20 44
C253 45 43 49 4D
C257 41 4C 20 43
C25B 4F 4E 56 45
C25F 52 53 49 4F
C263 4E 20 2D 20
C267 45 4E 54 45          FCC   "ENTER HEXIDECIMAL NUMBER"
C26B 52 20 48 45
C26F 58 49 44 45
C273 43 49 4D 41
C277 4C 20 4E 55
C27B 4D 42 45 52
C27F 04                   FCB   4
C280 41 4C 57 41  MSG2    FCC   "ALWAYS ENTER 4 DIGITS"
C284 59 53 20 45
C288 4E 54 45 52
C28C 20 34 20 44
C290 49 47 49 54
C294 53
C295 04                   FCB   4
C296 54 48 45 20  MSG3    FCC   "THE DECIMAL EQUIVALENT IS "
C29A 44 45 43 49
C29E 4D 41 4C 20
C2A2 45 51 55 49
C2A6 56 41 4C 45
C2AA 4E 54 20 49
C2AE 53 20
C2B0 04                   FCB   4
C2B1 45 4E 54 45  MSG4    FCC   "ENTER 'CR' TO QUIT OR SPACE"
C2B5 52 20 27 43
C2B9 52 27 20 54
C2BD 4F 20 51 55
C2C1 49 54 20 4F
C2C5 52 20 53 50
C2C9 41 43 45
C2CC 20 54 4F 20          FCC   " TO CONTINUE"
C2D0 43 4F 4E 54
C2D4 49 4E 55 45
C2D8 04                   FCB   4

                          END   START
```

Bram Frank, vice president
Intrigue/Compubyte
285 Dufferin Road
Montreal, Canada

(514)487-7111

February 2, 1983

We have owned Southwest Technical Products equipment since 1975 and have been running FLEX (its predecessors previous to that) since 1978. In 1980 we added an S/09 mainframe to the system and rebought most of our FLEX software.

The original system has run flawlessly for all these years with the singular exception of a pair of diodes in the original mainframe which flamed when we hit 56K of high current static ram. To be fair, Southwest warned us not to try it, but I did anyway.

In July 1981, I decided to 'fool around' with UNIFLEX. We had enough memory and also the CDS-1 Hard disk, so I placed the order. About three weeks later a package arrived through Canadian Customs and UNIFLEX was in.

We quickly brought up the system and it worked beyond our expectations.

However, with only two in-house production systems, our client base operating under FLEX and four programmers to share the systems, we required that both systems be running under FLEX. What a pain!! Opening the mainframe, rejumpering the disk controller, changing the monitor and moving the console I/O from $E000 to $E004 each time.

I subsequently discovered that the rejumpering was not required.

A partial solution was arrived at in August 1982. A switch was installed to allow us to select either S-BUG or UNIBUG. I took apart the S-BUG rom and re-addressed the I/O vector to $E000, patched most of our software which didn't go through FLEX for I/O (not much of that) and installed a switch for selecting monitor roms. It worked and worked well. But U.C.S.D. Pascal, Adventure and some other minor (to us) software did not work, not to mention that I would be at the mercy of all future software for patching.

One of the programmers suggested that perhaps it would be an idea, since UNIFLEX software has no direct access to hardware, to patch UNIFLEX for console operation at $E004 rather than all of our FLEX software.

Using DYNAMITE, I found the I/O vectors for the UNIBUG rom and then re-burned it. This brought up the monitor message on 'tty01'. Then I interchanged the names of 'tty00' and 'tty01'. It worked - except that the herald message still came out at $E000 and our tasks were wrongly identified as being run through 'tty01', while all other utilities pointed at 'tty00'.

The operating system has some fairly sophisticated protection against making a 'copy of a copy' of the UNIFLEX program, so we have built the following single byte patch into our 'crdisk' utility. The patch is written in basic for ease of implimentation and is inserted into the 'crdisk' utility just after the 'copy uniflexe /usr2' line. It is valid for UNIFLEX version 1.06.

```
10 open "/usr2/uolflex" as 1
20 position #1, 15641, mode 0
30 print #1, chr$(6);
40 close 1
50 sleep 10
60 exit
```

This byte and the one previous to it point to the port address where the herald message is written.

One other problem remained, since renaming the I/O drivers was unacceptable, the '/etc/init' program was patched to bring up 'tty01' as the console instead of 'tty00'. This patch was done on the supplied UNIFLEX master disc.

```
10 open "/etc/init" as 1
20 position #1, 707, mode 0
30 print #1, "1";
40 close 1
50 exit
```

This specifies that 'tty01' will be brought up when the single user mode is entered.

Thus, reburning the UNIFLEX monitor rom as described below, installing a switch to select monitors and running the above programs will allow for $E004 to be used as the console for both FLEX and UNIFLEX as well as allowing for a dual operating system environment, selectable at the flick of a switch and the push of a reset button.

Patch Table For UNIBUG rom

Addr from		to
0630	00	04
0636	00	04
063C	01	05
0665	00	04
066E	01	05
0676	00	04
067E	00	04

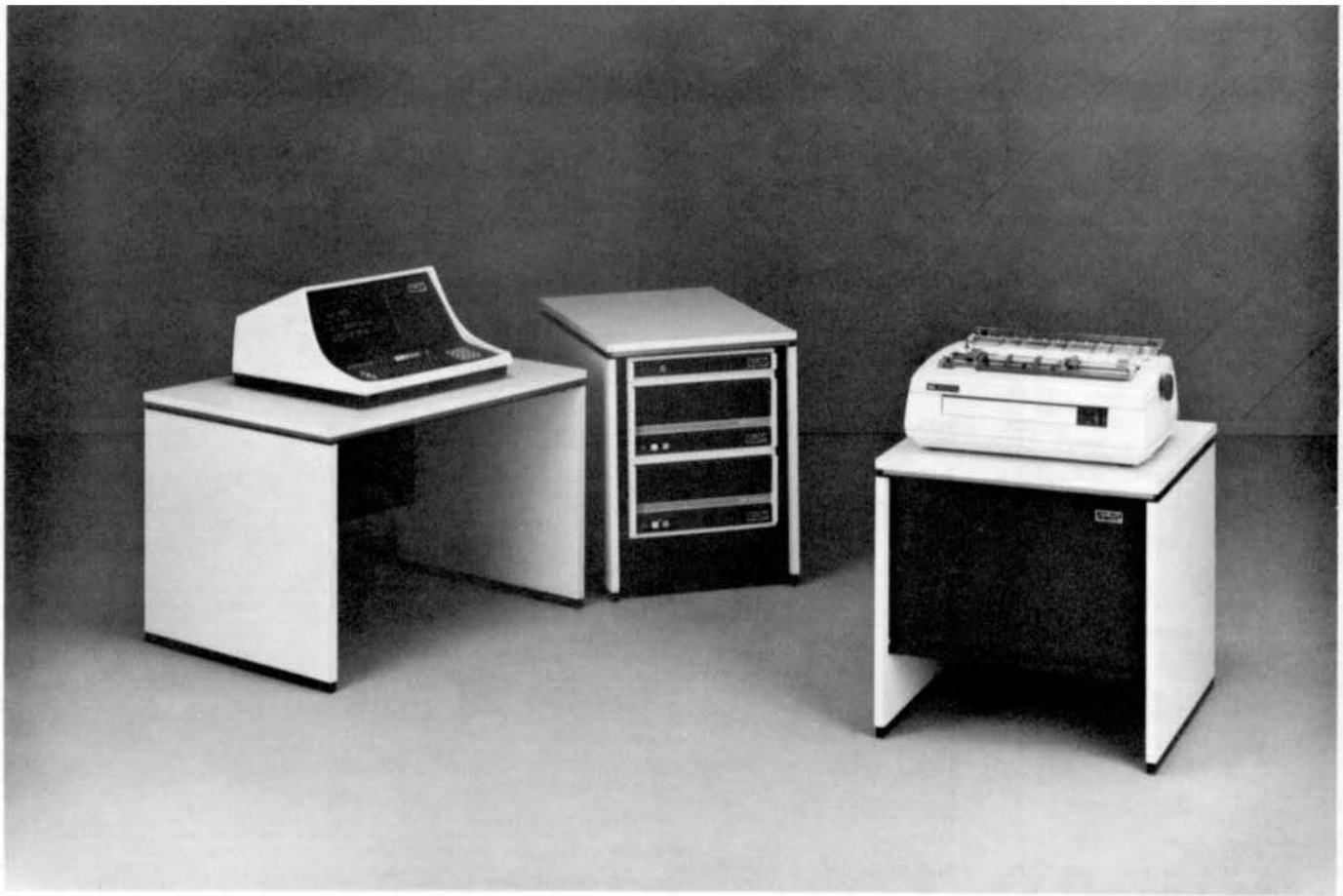
Addresses are in hex, offset from '$0000' in the rom.

Should enough interest be generated in the modification to allow for selecting monitor roms, I would be happy to generate another letter in this regard.

By the way Don, in the february issue you asked for reviewers for UNIFLEX software. I wouldn't mind being one of those people.

Sincerely

Bram Frank



THE COMPLETE BUSINESS SYSTEM

+Multiuser+Highly Expandable+Cost Effective

S+ THE CONCEPT

The S+ system is a modular computer system in which all portions of the hardware and software are designed to work together in the most efficient way possible. An S+ single user system with floppy disk storage is a competitive and cost effective entry level system. Unlike most other small computers being sold as "personal", or "small business" machines, the S+ system may be expanded to maximum capabilities using this same hardware and software. You cannot end up with a DEAD END system that cannot be expanded and whose software is not compatible with larger machines. A basic S+ system may be expanded to thirty-two users, a megabyte of main memory and hundreds of megabytes of hard disk storage by simply plugging in, or connecting the desired upgrade equipment.

TOTAL DESIGN—Hardware and Software

The S+ system is an integrated hardware and software design. The two complement and enhance each other in this system. The UniFLEX® operating system used in the S+ systems is patterned after the Bell Laboratories UNIX® operating system, one of the most admired and widely used operating systems in the world. Instead of being an afterthought, the software is part of the design of the S+ system. You can be sure that with this approach that all parts of the computer operate with maximum efficiency and cost effectiveness.

THE CENTRAL PROCESSOR

The basic S+ system is configured with 256K bytes of memory and can be expanded to more than 1 million bytes. An efficient and fast hardware memory management system is used to allocate the available memory among the users on a dynamic basis. As little as 8K bytes, or the entire memory—if needed—can be used by any individual user. This makes it possible to run very large programs on the system, but it also uses no more memory than necessary for a particular job. The increase in cost effectiveness of this system over crude and outdated bank switching arrangements is dramatic.

The central processor runs in both user and supervisor states. It can detect and reject a defective user program. It is impossible for a user program to go bad and stop the entire system, as can happen quite easily in less sophisticated systems.

Task switching is accomplished by use of a multiple map RAM memory, with sixty-four individual task maps. Each task can access from 4 to 64 K-bytes of memory. Multiple tasks may be used in programs that require more than 64K bytes of memory for execution. When a task is completed the memory is automatically released for other use.

SOFTWARE

The S+ operating system, UniFLEX® is a multiuser, multitasking operating system based on the UNIX® operating system that has been used for many years on Digital Equipment Corp. PDP-11 series minicomputers. It is considered one of the most sophisticated and "user friendly" operating systems available. Variations of UNIX® are rapidly becoming standard on mini and larger microcomputers.

A large variety of languages are available for use with the system. These include FORTRAN, COBOL, BASIC, and Pascal. Word processing packages are also available to give you full text processing capability on the system.

Applications programs are available in large quantities in many fields. This includes general business, medical, dental, veterinary, library and real estate management; plus others. Since the system is multiuser it can also be connected to cash registers to produce a point-ofsale terminal system combined with the computer. The possibilities for application of this system are endless.

THE I/O SYSTEM

The S+ system is totally interrupt driven. All terminal and printer I/O devices connect to an I/O bus separate from the main bus. Up to thirty-two separate devices may be connected to the I/O bus at any one time. If I/O activity is great enough to cause an unacceptable slowdown in system operation, a separate I/O processor can be installed in the system. This plug-in option removes all I/O handling overhead from the main processor and allows operation of up to thirty-two external devices at 9,600 baud. Without an integrated total design, as in the S+ system, it would become impractical to use a UNIX®type operating system in a situation with heavy terminal I/O activity.

DISK STORAGE

A wide range of disk storage capacity is available for the S+ system, from 2.5 M-byte floppy disks to an 80 M-byte Winchester and many sizes between. All disk controllers use direct memory access (DMA) type operations to maximize data transfer and to minimize overhead on the main processor. The Winchester disks also use intelligent controllers along with DMA transfers to preserve the performance that these type devices are capable of giving. Without this distributed intelligence the system performance would be greatly degraded. The UniFLEX®operating system is designed to work at maximum efficiency with this type disk system. The data transfer rates achieved by this combination rival those of large minicomputers.

COMMUNICATIONS

A high speed local network communications system is available to interconnect S+ systems. The VIABUS® network will allow communication between systems at data rates of over 400K baud. Such a system makes it possible to share data between local systems in an efficient and low-cost manner.

AVAILABLE SOON

Tape backup—20M-Byte in less than 15 minutes on a standard ¼ inch cartridge.

Mini-Wini—5 and 10 M-Byte Winchesters—5¼ inch package. Winchester performance, for smaller systems in a small package. UniFLEX® compatible design.

Large Capacity—190 and 340 M-Byte Winchesters, plus SMD cartridge drives.

UniFLEX is a registered trademark of Technical Systems Consultants, Inc.

UNIX is a registered trademark of Bell Labs.

VIABUS is a registered trademark of Southwest Technical Products Corporation.

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 **(512) 344-0241**

PRACTICAL APPLICATION of a MICROCOMPUTER REQUIRING > 64K MEMORY

William A. Comer
Rt 1 Box 437-C
Catlettsburg, Ky 41129
606-739-5430 HOME 606-329-3333 EX 8875 WORK

The purpose of this article is to demonstrate a practical application of the EXTENDED MEMORY capability of a SWTP based 6809 CPU / FLEX DOS system. The following structure will be used:

A) LIST THE SYSTEM HARDWARE AND SOFTWARE CONFIGURATION
B) DEFINE THE PROBLEM OF USING THE SYSTEM AS A 56K NON-EXTENDED MEMORY CONFIGURATION.
C) COMPARE EXECUTION TIMES WITH AND WITHOUT USE OF EXTENDED MEMORY.

A) HARDWARE AND SOFTWARE CONFIGURATION:

a) CPU - SWTP MP-09B WITH IC-8 INSTALLED AND 1.25 MHZ CLOCK
b) MOTHER BOARD - ELECTRA
c) NON-EXTENDED 56K MEMORY - SOUTHEAST MICRO DM-64
d) EXTENDED MEMORY BOARD #1 - THOMAS INSTR S-R/R 48K
e) EXTENDED MEMORY BOARD #2 - DIGITAL SERVICE & DESIGN DSD M/R-32K
f) DISK CONTROLLER - GIMIX 5/8 WITH 1771
g) DISK - SIEMENS FDD100-5 SINGLE SIDE SINGLE DENSITY
h) DOS - GIMIX FLEX 1.3
i) EXTENDED MEMORY DRIVER - VDISK 1.3

B) PROBLEM DEFINITION

THE DATA FILES BEING MAINTAINED WERE 130+ DISK SECTORS IN SIZE WITH 600-800 RECORDS EACH DIVIDED INTO 10 SEPERATED FIELDS. THE AMOUNT OF MEMORY AVAILABLE FOR EDITOR DATA IN A 56K SYSTEM WILL NOT ALLOW PLACING ALL THE DATA IN MEMORY AT ONE TIME. THIS CAN BE A REAL PROBLEM IF YOU ARE CONSTANTLY USING THE "NEW" FEATURE OF THE FLEX EDITOR TO EXAMINE THE PORTION OF DATA LEFT ON THE DISK AS I WAS. THIS TRANSFER OPERATION TOOK 1 MINUTE AND 30 SECONDS !

THE SECOND MAJOR PROBLEM WAS THAT I WAS USING TSC SORT/MERGE TO PERFORM A LARGE NUMBER OF SORTS ON ALL DATA FILES. A TYPICAL SORT PROCEDURE TOOK 6 to 8 MINUTES.

C) EXECUTION TIME COMPARISONS:

TIME COMPARISONS WERE TAKEN ON MANIPULATION OF A DATA FILE 144 DISK SECTORS IN LENGTH CONTAINING 651 RECORDS THAT WERE DIVIDED INTO 10 SEPERATE FIELDS.

OPERATION	56K SYSTEM	VIRTUAL DISK 80K of EXT MEMORY
EXAMINE LAST RECORD	1 MIN 40 SEC	16 SEC
SORT DATA FILE	6 MIN 30 SEC	35 SEC
EDITOR SAVE OPERTION	1 MIN 23 SEC	12 SEC
COPY FILE FROM-TO: DISK DRIVE TO-FROM: VIRTUAL DISK		1 MIN 30 SEC

GEORGIA INSTITUTE OF TECHNOLOGY

SCHOOL OF BIOLOGY · ATLANTA, GEORGIA 30332 · (404) 894-3735/894-3700

Dear Mr. Williams:

One of the problems with the S-50 bus is the lack of specialized interfaces. Do you know of any sources for a video camera or IEEE-488 interface to the S-50 bus. I have not seen a video camera interface advertised and may have to go to an S-100 bus for my next laboratory computer, although I like my GIMIX with OS-9 and Basic 09.

Hope you can help me.

Best regards,

David B. Dusenbery
Associate Professor

```
* THIS PATCH IS DESIGNED TO OVERCOME THE PROBLEM
* WITH STYLO AS HIGHLIGHTED IN PETER STARK'S ARTICLE
* IN THE APRIL, 1982, ISSUE OF THE 68 MICRO JOURNAL

        D410  DRTAB   EQU   $D410    FLEX DRIVE TABLE

0CE1                  ORG   $0CE1
0CE1 BD   BE70        JSR   PATCH    INSTEAD OF OPENING, PATCH
0CE4 12               NOP

* THERE ARE SEVERAL BYTES AVAILABLE
* AT $BE70 IN MY SYSTEM

BE70                  ORG   $BE70    AVAILABLE AREA

  70 E6   03   PATCH  LDB   3,X      GET DRIVE NUMBER
BE72 58               ASLB           MULTIPLY BY TWO
BE73 EB   03          ADDB  3,X      MULTIPLY BY THREE
BE75 58               ASLB           CALCULATE SIX TIMES THE DRIVE NUMB
BE76 34   10          PSHS  X        SAVE X REGISTER
BE78 8E   D410        LDX   #DRTAB   POINT TO DRIVE TABLE
BE7B 5D               TSTB           CHECK TO SEE IF DRIVE IS  ZERO
BE7C 27   03          BEQ   PATCH1
BE7E 3A               ABX            ADD B TO X
BE7F 6F   84   PATCH1 CLR   0,X
BE81 35   10          PULS  X        RESTORE POINTER
BE83 86   02          LDA   #2       OPEN FOR WRITE
BE85 A7   84          STA   0,X
  87 39               RTS

                      END

0 ERROR(S) DETECTED

SYMBOL TABLE:

DRTAB  D410   PATCH  BE70   PATCH1 BE7F
```

COMPUTER SYSTEMS CENTER
OF ST. LOUIS INC.
13461 OLIVE BLVD.
CHESTERFIELD, MO. 63017
(314) 576-5020

February 10, 1983

PRESS RELEASE

Effective immediately, DYNACALC, our popular Electronic Spread-Sheet Program for 6809 computers, is available for OS-9 systems.

The FLEX and UniFLEX versions of DYNACALC have been acclaimed by hundreds of users since their introduction last summer as THE most useful piece of software ever produced for the 6809 microprocessor.

Now users of OS-9 (trademark of Microware and Motorola) systems, both level one and level two, can gain all the benefits made possible by DYNACALC.

With DYNACALC, computer USERS (not just programmers) can do just about any kind of computer operation, from simple invoicing to complex general ledgers. No programming skills are required to use DYNACALC!

Even if you are an experienced programmer, you still gain by using DYNACALC, as most jobs can be done in as little as ten percent of the time required to write an equivalent program in BASIC.

Those of you who sell 6809 computers will benefit by demonstrating DYNACALC with your hardware. DYNACALC is so easy to use that your equipment will really shine!

DYNACALC is similar in operation to VisiCalc (trademark of VisiCorp), which runs on most personal computers. VisiCalc has sold hundreds of thousands of small computers for Apple, IBM, etc. Now DYNACALC, considerably more powerful than VisiCalc, sells 6809 hardware.

DYNACALC for OS-9 is priced at only $250.00 per single-CPU license (specify 5" or 8" diskette), and is available from CSC at the above address, or from your local DYNACALC dealer.

Dealers, please inquire about selling DYNACALC in your store.

Mr. Don Williams, Sr.
'68' Microjournal
P.O. Box 849
Hixson, TN 37343

Dear Don,

I would like to thank you for taking up the cudgels on my behalf regarding the difficulties I was having with the Logical Devices 6809 CPU card.

Not wishing to appear out of character, I am quite pleased with the 6805 Simulator that Computer Systems Consultants sells. It has proven very useful in debugging 6805 programs before they are run in the actual system. Since most computer program documentation is a mystery to me, I called Dr. Pass for help regarding his software, and he was most helpful and patient.

Sincerely,

Jeffrey M. Craig
Apt. 912 - 3001 S. King Dr.
Chicago, IL 60616

word's worth

P.O. Box 28954
Dallas, Texas 75228
(214) 321-9285

9 January 83

Larry Williams, Executive Editor
'68' Micro Journal
5900 Cassandra Smith
Box 849
Hixson, Tennessee 37343

RE: A NEW 6809 SMALL-C COMPILER

I have just recieved a copy of a 6809 Small-C compiler from Dan Everhart in Washington. It looks pretty good, and the documentation is well-done, except that his photocopier wasn't quite up to par. Although it doesn't have

all of the features of Middle-C, it is still quite adequate.

The really neat thing about this compiler is that it INCLUDES SOURCE CODE AND IS IN THE PUBLIC DOMAIN. Dan has very generously offered to share his work with the rest of the 6809 community.

In order to get around the documentation problem, I have arranged to get the document on disk, and I will probably distribute it either on a separate disk or have it typeset.

I talked at some length with Dan to make absolutely sure he didn't mind me distributing it, and he told me that all he wanted was credit for the adaptation — so, here goes: I will be distributing this compiler package (I've dubbed it FDS-C for Public Domain Small-C) for $25, including choice of separate disk or hardcopy for the manual. Please specify disk size. Make check or money order out to Word's Worth. I take MC and Visa, but a $1 handling charge will be added to charge card orders for this package.

FDS-C was written to use the SWTPC optimizing assembler ASM09. I have not yet had the time to adapt it for TSC's assembler, our new relocating assembler, or RLOAD, but I definitely plan to. Mr. Everhart is an experienced Unix user, and his run-time library definitely has the flavor of the Unix environment. I expect it to be a real joy to work with.

For further information on this or other programs in the Word's Worth Public Domain Library, interested persons should send a #10 SASE to the above address. I use an answering service, so an SASE is actually more dependable than a phone call. I don't know what it's like elsewhere, but a good answering service is hard to come by around here! Also, due the volume of mail I get, inquiries without SASE's are sometimes subject to considerable delay.

Yours truly,

Howard Lee Harkness

Howard Lee Harkness

Pakistan's present investment climate was pivotal in the decision to proceed with the joint venture. As Ric Hammond, President of Smoke Signal, states, "The demonstrated progress of the country under the government of President Zia-ul-Haq and the government's active encouragement of private industry makes Pakistan the preferred choice for location in this region. Also, the United States government encourages investment in Pakistan with investment insurance provided through the Overseas Private Investment Corporation."

Pakistan Computers Limited is currently setting up offices throughout Pakistan. Mr. Mushtaq Ahmad, the Group Managing Director of Madco Electronics, has also developed a marketing program for Saudi Arabia and other Gulf countries. Because the computer software will operate in Urdu and Arabic languages, as well as in English, the computers will be adaptable for use in varied business regions with minimal training.

FOR FURTHER INFORMATION, PLEASE CONTACT:

Mr. Ric Hammond
President
Smoke Signal
31336 Via Colinas
Westlake Village, CA 91362
tel: (213) 889-9340

Mr. Masroor Elahi Khan
Managing Director
Madco Electronics, Ltd.
201 Ual Towers
I. I. Chundriger Road
P.O. Box 3129
Karachi, Pakistan
tel: 201106

electronic ideas inc.

6809 CASSETTE BASED SYSTEMS USERS

Yes, I'm still using a cassette based system, and I'm sure there must be others out there. I'd like to hear from the rest of you, especially about where you're finding software. I'd like to do the following: Send me short descriptions of software you've found available (adventure games, word processing, editors, assemblers, languages, whatever), along with the suppliers address and cost. Also send an S.A.S.E. I'll collect the descriptions and run off a mini "catalog" of sorts. Sound like a good idea? Write me at:

Mike Johnshoy
707 Continental Circle, #1213
Mt View, CA 94040
(415) 967-2048

SMOKE SIGNAL
31336 Via Colinas
Westlake Village, California 91362

news release

FOR IMMEDIATE RELEASE

AMERICAN, PAKISTANI FIRMS FORM JOINT VENTURE

Smoke Signal Broadcasting, Inc., of Westlake Village, California, and Madco Electronics, Ltd., of Karachi, Pakistan announce the formation of Pakistan Computers Limited--a Joint-Venture Company. The new company will provide and develop special customer-oriented software, operating in the Urdu, Arabic and English languages.

Smoke Signal manufactures the Chieftain Series of Business Computers which rated highest in performance among all tested computers priced below $35,000 in independently conducted evaluations by the University of Colorado.

Pakistan Computers Limited has been licensed, for an undisclosed amount, to manufacture the Chieftain Computers in Pakistan and will market these products in Pakistan and the Middle East. The new company estimates that indigenous labor and components will account to more than 30% of the manufacturing value added. Thus, when export sales reach 50% of production, a positive impact will be made on Pakistan's balance of trade.

Mr. Masroor Elahi Khan, Managing Director of Pakistan Computers Limited, who pioneered the development of Pakistan television, notes, "Just as I was among the first to bring television technology into Pakistan, I am proud to be the first to bring computer manufacturing into this country. In addition to normal business activity, our company will bring low-cost computers to schools, colleges and universities which presently cannot afford the high cost of imported computers."

BOX 448 LAKEFIELD ONTARIO CANADA K0L 2H0 TELEPHONE 705-743-9811

83 01 24

Mr. Don Williams Sr.
Micro 68 Journal
P O Box 849
Nixson TN
USA 37343

Dear Don:

In response to your appeal a couple of issues ago for information re potential problems that 68XX users may encounter, I am writing to point out that according to the MC6883 data sheet, this chip in the Color Computer does not refresh the dynamic ram when it's in the unconditional " fast clock " mode.

The machine may appear to run OK, because the action of the progam counter tends to put sequential addresses on the bus, and the refresh interval spec is usually quite conservative. Some software, notably a tight loop which waits a few hundred milliseconds for an input bit to change state, will probably cause loss of data in memory.

I don't have a CC to try this out on, but an easy way to see if it's really a problem would be to write a machine code delay loop that MUL's a 2**16 times and run it as a USR function from a double speed Basic program; if I'm right the machine will crash with the memory half full of garbage upon the RTS from the USR function.

Yours truly;

John Scott

John Scott

CAMBRIDGE ENGINEERING

P.O. Box 66 - Cambridge, VT 05444

I have just received version 1.4 of the INTROL C-Compiler, and have found a good product made better. In recompiling two existing programs which were originally compiled using INTROL version 1.3, I have realized reductions of 14% and 20% in memory requirements. The 20% reduction was realized in a program which uses extensive pointer structures, in particular, pointers to pointers inside of structures. In addition to improving the efficiency of the compiler, they

have now added longs and floats, making the software an excellent value! If you consider that another vendor is selling the IEEE floating point arithmetic package separately for several hundred dollars over the cost of the complete INTROL C-Compiler.

Sincerely,

Martin C. Poppe, Jr.
CAMBRIDGE ENGINEERING

SWTP Computer Systems

Southwest Technical Products

12 Trawlers Road
Orton Southgate
Peterborough PE2 0BG
Tel Peterborough (0733) 234423
Telex 32600

Mr D N W Williams
68' Micro Journal
5900 Cassandra Smith
P O Box 849
Hixson
TN 37343
USA

Dear Sir,

FLEX, although still a very worthy Operating System, is not a good enviroment for UNIX based software. This is easily demonstrated by the running of a typical benchmark on the UniFLEX machine using the SWTPc C Compiler.

Compiler	Execution Time	Total Bytes
Duggers C V2.1	24.6 sec	15864
Telcom C	31.2 sec	5613
SWTPc C	6.0 sec	3608

These are the figures for the random number benchmark that appeared on page 20 of the September '82 issue. As you can see there is really no basis for comparison with the SWTPc/UniFLEX combine being a different ballpark altogether, in fact the SWTPc/UniFLEX combination comes closer to competing with some very respectable (and hugely expensive) minis.

Regards

Russell Brown

SPECTRUM PROJECTS
93-15 86 DRIVE
WOODHAVEN, NEW YORK 11421

VOICE LINE (212)441-2907
DATA LINES (212)441-3755/3766

64K Upgrade for TDP100, 26-3002A, and 26-3004A

1. Remove capacitors C58, C60, C62, C64, C66, C68, C70, and C72.

2. Move two jumpers to the left of U21 and one jumper above U28 down to the 64K position.

3. Solder the two bare staking pins to the left of U17 together.

4. Install 64K chips.

AVAILABLE FROM SPECTRUM PROJECTS

16K Chips	$16/SET
64K Chips	$64/SET
BASIC 1.1 ROM	$36.00
6883/HEATSINK	$29.95
ABOVE	$29.95
EXTENDED BASIC KIT	$88.00

FOR QUICK SERVICE, CALL (212)441-2907

ERNEST STEVE WATSON
11701 ST. CHARLES BLVD.
LITTLE ROCK, ARKANSAS 72211
501/224-1158

Re: Home Accounting Program System

Dear Sir:

When Dale Brady and I sent you MAPSYS, we indicated that some needed programs had not been completed and invited your readers to respond. Mr. Stig Soberg of Linkoping, Sweden did so and I am enclosing a copy of his program which "clears" all general ledger accounts at year end. Dale and I had indicated a need for a similar program when the first installment of MAPSYS was published (July, 1982 issue of 6809).

Without disparagement to Mr. Soberg's program, his work encouraged us to write the enclosed program (CLEAR.SAS), which "clears" only the income and expense accounts and posts the difference to the Equity Account (Acct. No. 300 in our systems). This is the way in which accountants complete their year-end entries and your readers might like to have it for that purpose.

Thanks again for your excellent publication.

Sincerely,

```
10 REM CLEAR.BAS
20 REM THIS PROGRAM WILL CLEAR ALL INCOME AND EXPENSE ACCOUNTS
30 REM PLUGGING EQUITY WITH THE DIFFERENCE
40 REM 1,28,83
50 PRINT"**************WARNING*****************"
60 PRINT
70 PRINT "THIS PROGRAM CLEARS ALL INCOME AND EXPENSE ACCOUNTS"
80 PRINT "AND SHOULD ONLY BE RUN AT YEAR END TO CREATE A NEW "
90 PRINT "'DEC.GL' GENERAL LEDGER FILE FOR THE NEXT YEAR."
100 PRINT:PRINT
110 PRINT "THE ORIGINAL 'DEC.GL' SHOULD BE PRESERVED."
120 PRIN :PRINT
130 PRINT "DO YOU WANT TO CONTINUE (Y/N)?";AN$=INCH$(0)
140 IF AN$<>"Y" THEN 580
150 REM READ IN END OF YEAR GENERAL LEDGER
160 OPEN OLD "DEC.GL" AS 1
170 GET #1, RECORD 1
180 REM X%=NUMBER OF ACCOUNTS
190 FIELD #1,2ASZ$:X%=CVT%%(Z$)
200 REM N%=NO.;A$=ACCT.NAME;A=ACCT.BAL.
210 DIM N%(X%),A$(X%),A(X%)
220 GET #1,RECORD 1:GOTO 240
230 GET #1
240 FOR S%=0 TO 7
250 FIELD #1,S%*30AS$,2ASGN$,20ASGM$,8ASGT$
260 IF I%=0 THEN 320
270 N%(I%)=CVT%%(GN$):A$(I%)=GM$:A(I%)=CVTSF(GT$)
280 REM KEEP A RUNNING TOTAL OF INCOME
290 REM AND EXPENSE ACCTS. TOTALS
300 IF N%(I%)>399 THEN A1=A1+A(I%)
310 IF I%=X% THEN 350
320 I%=I%+1
330 NEXT S%
340 GOTO 230
350 CLOSE 1
360 ON ERROR GOTO 600
370 OPEN NEW "DEC.GL" AS 1
380 REM ACCT.300 IS EQUITY; INCOME & EXPENSE
400 REM ACCTS. START AT 400
410 R%=0
420 FORS%=0TO7
430 FIELD#1,S%*30AS$,2ASGN$,20ASGM$,8ASGT$
440 IF N%(R%+S%)=300 THEN A(R%+S%)=A(R%+S%)+A1
450 IF N%(R%+S%)>399 THEN A(R%+S%)=0
460 LSETGN$=CVT%$(N%(R%+S%))
470 LSETGM$=A$(R%+S%)
480 LSETGT$=CVTF$(A(R%+S%))
490 IFR%+S%=X%THENPUT#1:GOTO530
500 NEXTS%
510 PUT#1
520 R%=R%+8:GOTO420
530 GET#1,RECORD1:REM ADD FILE SIZE
540 FIELD#1,2AS2$
550 LSETZ$=CVT%$(X%)
560 PUT#1,RECORD1
570 CLOSE 1
580 END
590 ONERRORGOTO
600 IFE R=77THENRESUME530
```

technical systems consultants, inc.

111 Providence Road • Chapel Hill, North Carolina 27514 • (919) 493-1451 • TWX 510-920-0540

NEW PRODUCT BULLETIN
Release date: IMMEDIATELY
Contact: Dan Vanda
Phone: (919) 493-1451

NEW 6809 COBOL COMPILER

Technical Systems Consultants, Inc. has announced the release of a native-code COBOL compiler for use under their 6809 microprocessor-based UniFLEX™ Operating System. The product was developed totally in-house and has undergone considerable field testing. Based on the ANSI 1974 COBOL standard, the compiler offers nearly all Level I and many Level II

features. Because the operating system, UniFLEX™, is multi-user (very similar to UNIX™), several unique features have been built into this COBOL. Of particular interest, the "TERMCAP" facility, in harmony with UniFLEX™, enables any COBOL program to run - without modification - on any cable of terminal. The COBOL programmer is relieved of any terminal specifics. His program will automatically handle the cursor for screen I/O at any terminal, regardless of its characteristics.

A must for any multi-user environment, record locking is supported for disk files. A "shared text" option produces COBOL programs that require only one copy of the program in memory even though it is simultaneously run by multiple users. An interactive debug option permits a user to trace the execution of a program, insert breakpoints, single-step through a program, and more. A unique feature allows a user to interact with the debugger on one terminal while any program I/O is carried out on a second terminal. This is particulary convenient when the program under test performs screen oriented I/O which would be destroyed by debug prompts and input on the same terminal.

The SCREEN SECTION for cursor oriented input/output to a CRT terminal is one of the most complete in the industry. It supports all the screen protocol options offered by both Data General and NCR COBOLs. Key capabilities include direct and relative cursor positioning, highlighting, screen and line erase, and extended formatting of display and accept fields. Screen fields may be specified as right justified, required, secure, full, auto, and blank when zero.

Disk file input/output may be sequential, indexed, and true relative. Variable length records are allowed in any of these file types. The indexed file type allows up to four alternate keys per file, with or without duplicates. Except for specifying an alternate COLLATING SEQUENCE, all of the Level II SORT-MERGE features are implemented as well as a complete Level I Report Writer package. Also implemented is an extended CALL verb which is not limited to calling COBOL programs; any program executable under UniFLEX™ may be called. Users familiar with "pipes" under UniFLEX™ or UNIX™ will appreciate the ability of this COBOL to generate a pipe.
Numerous compile-time options allow the production of source listing, cross reference listing, and variable summary information, and the specification of output filename. A single call to the compiler will produce an executable binary program, but it is possible to halt the compilation at the intermediate assembler source stage if desired.

A single-cpu license for the complete 6809 UniFLEX™ COBOL package costs $750.00, which includes one year of maintenance and a limited warranty, and may be obtained from Technical Systems Consultants, Inc., 111 Providence Road, Chapel Hill, NC 27514. Telephone: (919) 493-1451, Telex II: 510-928-0340.

™ UniFLEX is a trademark of Technical Systems Consultants, Inc.
™ UNIX is a trademark of Bell Laboratories.

UNIVERSAL DATA RESEARCH, INC. ANNOUNCES

A FULL LINE OF MODEMS AND THE ALL NEW

"SPEED SELECT" MODEM I/O CARD

Universal Data Research, Inc. today announced a new complete line of modems and an I/O card that automatically selects the appropriate baud rate. The new UDRI modems are designed to be simple to use, reliable, and priced below the average market.

The UDRI modems are designed to connect any RS232 interface terminal or computer with the phone lines. The 300 baud modems provide half and full duplex operation in originate, answer and auto-answer mode . The 1200 baud modems provide full duplex operation with switch selectable local echo. The digital design is based on two LSI integrated circuits and is crystal controlled for superior performance.
The 212A modems are compatible with the Bell 212A modem. They automatically dial/answer and transmit at 300/1200 baud. Unlike most others, these modems do not require any special wiring or interfacing. These modems can be used with the UDRI automatic speed select I/O board.

All UDRI modems are attractively housed in a compact, brushed aluminum, anodized case which neatly fits under a standard desktop phone. With low parts count, these new modems have outstanding reliability, long life, and lower costs - which are reflected in a lower price.

Prices for these modems are as follows:

300 baud acoustic	$149.00
300 baud direct	179.00
300 baud auto-answer	219.00
1200 baud direct	449.00
1200 baud auto-answer	499.00
212A auto-answer	549.00
212A auto dial	599.00

The all new modem I/O board may be used with the 212A modems to automatically select the baud rate. This I/O card allows the modem to test for 1200 baud and automatically adjust to 300 baud without requiring special changes in the hardware configuration. The card provides greater flexibility and is available only through UDRI.

The I/O speed select board is available for only $119.00. Dealer inquiries are welcome.

For further information contact:

UNIVERSAL DATA RESEARCH, INC.
Cynthia Dyla Wilson
Director of Marketing
2457 Wehrle Drive
Buffalo, NY 14221
(716) 631-1011

MICROWARE PRESS RELEASE

CONTACT: H. JEROME KAPLAN

Microware Systems Corporation announces the introduction of Support Service Registration Cards. The Support Service entitles the customer to use the Software Hot-Line or Telex Service for technical advice and consultation. The registration cards, when filled out and returned, entitle the customer to 90 days FREE Hot-Line support.

Microware and 11 other 6809 based computer manufacturers are distributing two types of registration cards. The cards are color coded, blue for the OS-9 operating system and brown for languages and software tools. Both types of cards give complete information about the Hot-Line service (hours, phone and telex number, etc.) and a customer serial number.

The Support Service Registration Cards are being distributed with all Microware software products shipped after January 15, 1983. Software support commences on the purchase date, however, the card must be filled out and immediately returned to Microware to initiate the free 90 day support period.

Microware offers the Yearly Support Service to customers, upon the termination of the free 90 day support period, for $75.00 per year.

N E W S R E L E A S E

Computerware™ introduces 64K SCREEN EXPANDER available on cassette and disk for the Radio Shack Color Computer and TRP-100.

The 64K SCREEN EXPANDER allows the 64K Color Computer to have a 51 x 24 upper and lower case display for everything! This includes Basic and all assembly language programs that use text displays. Included with it is a very nice character editor so if you want to change any of the characters, it is very easy to do so. What it does is transfer all your ROMs to RAM and then modify them to use its new Hi-Res display. From our testing, it does not affect any software, stays even after resetting, and looks great even on a TV.

The PRINT command is also expanded. It now works with true coordinate positions, (absolute cursor positioning). You simply give PRINT@ the Y and X coordinates of the position you want to print! The 64K SCREEN EXPANDER requires 64K of memory and Extended Basic.

THE 64K SCREEN EXPANDER only costs $24.95 on cassette and $29.95 on disk (plus $2 shipping and handling). It's available from COMPUTERWARE™ dealers or directly from COMPUTERWARE™ at Box 668, 4403 Manchester Ave., Suite 103, Encinitas, CA. 92024, (619)436-3512.

Box 668 • 4403 Manchester Ave. • Suite 102 • Encinitas, California 92024
Phone: (619) 436-3512

University of South Florida
Biology Dept., LIF 169
4202 Fowler Avenue
Tampa, FL 33620
(813) 974-2698
1 October 1982

Computer Publishing Center
68 Micro Journal
5900 Cassandra Smith
P.O. Box 849
Hixson, TN 37343

Dear Don:

I received your letter of September 17 and am glad that you decided to publish the material I submitted earlier. I am enclosing a disk containing material for a second article. It is a random number generator for UniFLEX FORTRAN written in assembler language and designed to be incorporated into a FORTRAN library by the method described in the first article. UniFLEX FORTRAN does not have an intrinsic random function, so this is a logical addition to one's utility library. Random number generation is a special interest of mine, and I wrote a simpler but less rigorous version for the 6502 that appeared in the August 1982 issue of MICRO (the other MICRO).

The disk is in UniFLEX format, single sided, single density. All five files are text files written with the UniFLEX text editor. I did the indentations and tabs with multiple spaces. The files are:

sethenv2	text of the article
seed	Listing 1
rnd	Listing 2
rndchi.f	Listing 3
rndses.f	Listing 4

I am also enclosing a listing of these files for verification.

Unless something unforeseen steals my time, I will send you my library of string-handling utilities in 2 or 3 weeks. This is another tool for FORTRAN users, but written in 6809 assembler language (and thus readable by everyone).

Art Mathenv
++

```
++ list seed
*                LISTING 1
*
*
*
* * *  SEED  * * *
*
* Seeds the Random Number Generator
*
* Should be called before 1st RND
*
* subroutine seed(I)
*
* if I = 0, uses time system call
* if I<> 0, uses I
*
 name _seed
 global _seed:nrnv
 text
_seed leax 2,s
 pshs d

* Is argument zero?

 ldd [,x]
 bne seed1

* Use argument to seed N

 std nrnv
 std nrnv+2
 std nrnv+4
 std nrnv+6
 puls d,pc

* Use time system call

seed1 sys 39,nrnv+2
 ldd nrnv+5
 std nrnv
 puls d,pc

* N array & time buffer

 bss

nrnv rmb 10

 end
++

 list rnd
*                LISTING 2
*
*
*
* * *  6809 Random Number Generator  * * *
*
* by Art Mathenv
*
* Uses Linear Consruential Operator:
*
*      N' = (ASN+C) mod P
*
* where: N is a 7-byte integer
*        P = 100000000000000 H
*        A =   1010101010101 H
*        C =  36BC6344F997A5 H
*
* The 7 bytes of N must be seeded once.
* The seed value determines the pseudo-
* random sequence.

 name _rnd
 ext nrnv
 global _rnd
 text
_rnd leax 4,s address to put result
 pshs d,x,u save registers

**********  Calculations  **********

* Perform linear consruential operation

 bsr rnd0

* Move result before renormalization

 bsr move
 bcc zero test for zero

* Renormalize result

 bsr norm

* Clean up the stack and return

 puls d,x,u,pc

* Once in a billion years...

zero lda #0 result is zero
 sta ,x
 puls d,x,u,pc return

**********  Subroutines  **********

* Linear Consuential Operator

* Multiply by A = 1010101010101 H

rnd0 ldu #nrnv+7 move u above least byte
 ldx #7 number of bytes in N
 ldd #0
rnd1 pshs a add a to b
 addb ,-u
 adca #0
 addb ,-u add next byte
 adca #0
 stb ,u replace this byte of N
 leax -1,x
 bne rnd1 next byte
```

```
* Renormalize result

norm lda #$80 set exponent to zero
 sta 7,x exponent location
norm1 lda ,x most significant byte
 anda #$80
 bne norm3
 ldb #7 shift decimal
 leay 7,x
 clc
norm2 rol 1,y
 decb
 bne norm2
 dec 7,x adjust exponent
 bra norm1

* Clear sign bit

norm3 lda ,x
 anda #$7F
 sta ,x
 rts

 end
++


* Add C = 36BC6344F997A5 H

 ldb 6,u
 addb #$A5
 stb 6,u
 ldb 5,u
 adcb #$97
 stb 5,u
 ldb 4,u
 adcb #$F9
 stb 4,u
 ldb 3,u
 adcb #$44
 stb 3,u
 ldb 2,u
 adcb #$63
 stb 2,u
 ldb 1,u
 adcb #$BC
 stb 1,u
 ldb ,u
 adcb #$36
 stb ,u
 rts

* Move M to result location
* u-reg must point to M

move tfr new u-reg points to result
 ldb #7
 clc Carry clear if zero
move1 lda ,u+
 sta ,y+
 bcc move2
 sec
move2 decb
 bne move1
 rts
   list rndchi.f
*               LISTING 3
*
*
*
* * *  RNDCHI  s * *
*
* Chi-Squere test of RND function
*
* compile: f77 rndchi.f +c
*
* link-edit:
*    rndchi.r
*    rnd.r
*    +l=F77.runlib
*    +n
*    +o=rndchi
*
      external rnd,seed
      real rnd
      integer knt(100)
      write(6,90)
   90 format('Give seed value: ',$)
      read(5,91) J
   91 format(I6)
      call seed(J)
      write(6,92)
   92 format(/'99 degrees of freedom'//
     +'PASSES   CHI-SQUARE')
      do 1 I=1,100
      knt(I)=0
    1 continue
      do 2 J=1,100
      do 3 I=1,100
      N=int(100*rnd(0))+1
      knt(N)=knt(N)+1
    3 continue
      C=0.
      do 4 I=1,100
      D=float(knt(I)-J)
      C=C+D*D
    4 continue
      write(6,93) J,C/float(J)
   93 format(I3,F12.2)
    2 continue
      stop
      end
```

```
   list rndsee.f
*               LISTING 4
*
*
*
* * *  RNDSEE  * * *
*
* Sequential correlation test of RND function
*
* compile: f77 rndsee.f +c
*
* link-edit:
*    rndsee.r
*    rnd.r
*    +l=F77.runlib
*    +n
*    +o=rndsee
*
      external rnd,seed
      character*1 ch
      real rnd,cross(100,12),free(12)
      integer digit(12),hist(100,12),seq
      data ch/'*'/
*
* Dialog
*
      write(6,1)
    1 format('Input sequence length (recommended: 1000) ',$)
      read(5,2) seq
    2 format(f12.0)
      if(seq.lt.2.) seq=2.
      if(seq.gt.9999.) seq=9999.
      nset=int(1.+alog(seq)/alog(2.))
      write(6,3) seq,nset
    3 format('Sequence length =',f8.0/
     +'Input maximum correlation offset (recommended:',I3,')'/
     +'Use digits only! ',$)
      read(5,4) nset
    4 format(I3)
      if(nset.gt.100) nset=100
      if(nset.lt.1) nset=1
      write(6,5) nset
    5 format('Max offset =',I4/
     +'Input seed value (integer): ',$)
      read(5,4) Jseed
      call seed(Jseed)
*
* Zero all sums
*
      count=1.
      do 6 J=1,12
      free(J)=0.
      do 6 i=1,nset
      cross(i,J)=0.
    6 continue
*
* Load up History array
*
      do 7 i=1,nset
      call bits(digit)
      do 7 J=1,12
      hist(i,J)=digit(J)
    7 continue
*
* Main Loop
*
   10 if(count-100.*float(int(count/100.)).lt.1.) write(6,9) ch
    9 format(a1,$)
      call bits(digit)
      do 11 J=1,12
      free(J)=free(J)+float(digit(J))
   11 continue
      do 12 i=1,nset
      do 12 J=1,12
      k=eor(hist(i,J),digit(J))
      cross(i,J)=cross(i,J)+float(k)
   12 continue
      i=nset
   13 if(i.eq.1) go to 15
      do 14 J=1,12
      hist(i,J)=hist(i-1,J)
   14 continue
      i=i-1
      go to 13
   15 do 16 J=1,12
      hist(1,J)=digit(J)
   16 continue
      count=count+1.
      if(count.lt.seq+.5) go to 10
*
* Print Results
*
      ave=seq/2.
      sigma=sqrt(seq)/2.
      write(6,20) seq,nset,Jseed,ave,sigma,free,
     +((i,cross(i,J),J=1,12),i=1,nset)
   20 format(////'Sequence length =',f8.0/
     +'Max offset =',I4/
     +'Seed value =',I4//
     +'Each entry below should be a binomial variate with'/
     +'   expectation value =',f7.0/
     +'   standard deviation =',f6.0//
     +26x,'Frequency'/
     +'digit',12f6.0//
     +'offset',20x,'correlation'/
     +(I4,12f6.0))
      stop
      end
```

```
*****************************************************************
*
      subroutine bits(digit)
      integer digit(12)
      x=rnd(0)
      do 1 i=1,12
      x=x*x
      j=int(x)
      x=x-float(j)
      digit(i)=j
    1 continue
      return
      end
*
* Exclusive OR function
*
      integer function eor(i,j)
      eor=i+j-2*i*j
      return
      end
++
   list mathenv2
```

A RANDOM NUMBER GENERATOR FOR UNIFLEX FORTRAN

by Art Mathenv
University of South Florida
Biology Dept., LIF 149
Tampa, FL 33620

The random number generator described in this paper is designed to work as a utility function for FORTRAN 77 as implemented under UniFLEX. However, any 6809 programmer can make use of the subroutine beginning at the 'rnd0' label of Listing 2.

The general procedure used here was described in an earlier article entitled 'How to Write Machine-Language Utilities for UniFLEX FORTRAN'. (UniFLEX is a trademark of Technical Systems Consultants.) Mathematically, the random numbers are generated by the application of a linear consuential operator. Such operators were described in detail by T. F. Elbert in a series of four articles published in this Journal from November 1981 to February 1982.

Listing 1 is a subroutine which must be called once before the random number generator (RNG for short) is used. It is called 'seed' because it seeds the RNG. If the argument is non-zero, then that value is used to seed the RNG; if it is zero then the 'time' system call (number 39) will be invoked for this purpose.

Listing 2 is the RNG itself. This is a real function, which is similar to an integer function such as 'necs' of the earlier paper. In both function types, the function program is expected to place the result at a certain point in the stack. The only difference is the type of the result. The result of an integer function must be a two-byte integer, while the result of a real function must be an eight-byte real number, in the format used by FORTRAN 77 for real values.

A real value consists of eight bytes. The first seven bytes are the mantissa, and the eighth byte is the exponent. The first byte is the most significant byte, and the seventh byte is the least significant byte. Bit seven (MSB) bit of byte 1 indicates the sign. I presume that the result must be 'renormalized'. By this I mean the following: If the result is not zero (in which case special handling is required), the most significant bit of the most significant byte must be a '1'. If it is not, then shift the mantissa left and adjust the exponent until a '1' is shifted into this position. Then replace that bit with the sign bit, but in our case, where the result is always positive, the sign bit is always '0'. I'm sorry if that's confusing, but I have just told you everything I know about floating-point arithmetic.

Since there are seven bytes in the mantissa of the result, the RNG uses a seven-byte integer, which is set up by 'seed' in the bss segment at 'nrsv'. (The extra three bytes are for the portion of the time buffer which is not used by the seed program.) The RNG generates a new seven-byte integer from the old one by applying the following linear consuential operator:

$$N' = (A*N+C) \bmod P$$

where: N is the old integer
N' is the new value
P = 100000000000000 H
A = 101010101010101 H
C = 36DC6344F997A5 H

The starting value is generated by 'seed', and the sequence is propagated by repeated application of this same operator. The starting value will not re-occur until all possible P numbers have been generated. (T.G. Newman and P.L. Odell, THE GENERATION OF RANDOM VARIATES, Hafner Publishing Company, New York, 1971, page 8.) At that point the sequence will repeat. (But you will not live long enough to see it!)

Because the period of this sequence is so large, you can generated a virtually unlimited number of random numbers with this function. In addition, it is pretty fast, first because it is in machine language,

and second because I took a sneaky short cut in multiplying by A. These are important considerations in applications such as Monte Carlo methods or statistical simulations.

The most obvious weakness in this method is the regular pattern of bits in the multiplicative constant A. Is this a fatal flaw? We can test it to find out.

Listings 3 and 4 not only illustrate how the RNG is implemented in FORTRAN programs, but also test the statistical quality of the RNG. Listing 3 is the chi-square test. Random integers between 1 and 100 are chosen, and a count is made of how many times each of these values is chosen. The chi-square statistic calculated from this data should be around 99. Anything near that value is acceptable because the chi-square of a bad RNG will diverge rapidly. A chi-square value that is too low (converging toward zero) is just as bad for some applications as one that is too high (diverging upward).

Listing 4 is another test program. It analyzes the first 12 significant bits of the generated values. It counts the number of '1' bits in the sequence at each of these digits. It also does a kind of autocorrelation test by counting the number of times a digit is the same as the corresponding digit of its predecessors. All of these statistics ought to have a simple binomial distribution. If all of them are within (Oh! let's say) five standard deviation units of the expected mean, then I would call it okay.

Looking at the results of these tests, the RNG looks fine to me. However an RNG that works great in one application might completely bomb in another. There is no absolute test for whether a random number generator "works" or not. Thus you may be well advised to run further tests to make sure that this RNG is suitable for your application.

PROCEDURE

1) Enter the source code for "seed" and "rnd" from Listings 1 and 2.
2) Assemble the routines with the relocating assembler:

```
++ relasmb rnd seed
```

This produces a relocatable object-code module and names it "rnd.r".

3) Compile the main program ("rndchi.f", for example) with the +c option:

```
++ f77 rndchi.f +c
```

This produces another relocatable object-code module: "rndchi.r".

4) Link the modules together with the "link-edit" command:

```
++ link-edit rndchi.r rnd.r +l=F77.runlib +no=rndchi
```

This produces an executable module. We have specified the name of the output file to be "rndchi". For large programs, you may wish to use the "+t" rather than the "+n" option. The "F77.runlib" library is the run-time library. I guess--anyway, you need it.

5) The executable module is now ready!

```
++ rndchi
```

This should set the program underway.

++

TEN MOST-ASKED QUESTIONS

ABOUT DYNACALC™

THE ELECTRONIC SPREAD-SHEET FOR 6809 COMPUTERS

1. **What is an electronic spread-sheet, anyway?**
   Business people use spread-sheets to organize columns and rows of figures. DYNACALC simulates the operation of a spread-sheet without the mess of paper and pencil. Of course, corrections and changes are a snap. Changing any entered value causes the whole spread-sheet to be re-calculated based on the new constants. This means that you can play, 'what if?' to your heart's content.

2. **Is DYNACALC just for accountants, then?**
   Not at all. DYNACALC can be used for just about any type of job. Not only numbers, but alphanumeric messages can be handled. Engineers and other technical users will love DYNACALC's sixteen-digit math and built-in scientific functions. There's even a built-in sort command, so you could use DYNACALC to manage small data bases - up to 256 records.

3. **What will DYNACALC do for ME?**
   That's a good question. Basically the answer is that DYNACALC will let your computer do just about anything you can imagine. Ask your friends who have VisiCalc, or a similar program, just how useful an electronic spread-sheet program can be for all types of household, business, engineering, and scientific applications.

4. **Do I have to learn computer programming?**
   NO! DYNACALC is designed to be used by non-programmers, but even a Ph.D. in Computer Science can understand it. Built-in HELP messages are provided for quick reference to operating instructions.

5. **Do I have to modify my system to use DYNACALC?**
   Nope. DYNACALC uses any standard 6809 configuration, so you don't have to spend money on another CPU board or waste time learning another operating system.

6. **Will DYNACALC read my existing data files?**
   You bet! DYNACALC has a beautifully simple method of reading and writing data files, so you can communicate both ways with other programs on your system, such as the Text Editor, Text Processor, Sort/Merge, RMS data base system, or other programs written in BASIC, C, PASCAL, FORTRAN, and so on.

7. **How fast is DYNACALC?**
   Very. Except for a few seldom-used commands, DYNACALC is memory-resident, so there is little disk I/O to slow things down. The whole data array (worksheet) is in memory, so access to any point is instantaneous. DYNACALC is 100% 6809 machine code for blistering speed.

8. **Is there a version of DYNACALC for MY system?**
   Probably. You need a 6809 computer (32k minimum) with FLEX or UniFLEX operating system. A version for OS-9 is also in the works. You also need a decent CRT terminal, one with at least 80 characters per line, and direct cursor addressing. If your terminal isn't smart enough for DYNACALC, you probably need a new one anyway. The UniFLEX version of DYNACALC also allows you to mix different brands of terminal on the same system. There's also a special version of DYNACALC for Color Computers equipped with FLEX and Data-Comp's F-MATE. A version for Frank Hogg's Color Computer FLEX is also being done.

9. **How much does DYNACALC cost?**
   The FLEX versions are just $200 per copy; UniFLEX version $395. Foreign orders add $10 per copy for postage. We encourage dealers to handle DYNACALC, since it's a product that sells instantly upon demonstration. Call or write on your company letterhead for more information.

10. **Where do I order DYNACALC?**
    See your local DYNACALC dealer, or order directly from CSC at the address below. We accept telephone orders from 10 a.m. to 6 p.m., Monday through Friday. Call us at 314-576-5020. Your VISA or MasterCard is welcome. Please specify diskette size for FLEX versions. Software serial number is required for the UniFLEX version of DYNACALC.

ORDER YOUR DYNACALC™ TODAY

Foreign Dealers:

Australia & Southeast Asia: order from Paris Radio Electronics, 7A Burton St., Darlinghurst, NSW 2010 Sydney. Telephone: 02-357-5111.

United Kingdom: order from Compusense, Ltd., PO Box 169, London N13 4HT. Telephone: 01-882-0681.

Scandinavia: order from Swedish Electronics hk AB, Murergatan 23-25, Uppsala S-754 37 Sweden. Telephone: 18-25-30-00.

Computer Systems Center
13461 Olive Blvd.
Chesterfield, MO 63017
(314) 576-5020

OS-9 Version NOW Available

$250.00

UniFLEX software prices include maintenance for the first year.

DYNACALC, DYNAMITE, and DYNAMITE + are trademarks of Computer Systems Center.

F-MATE is a trademark of Data-Comp.
VisiCalc is a trademark of VisiCorp.
OS-9 is a trademark of Microware and Motorola.
FLEX and UniFLEX are trademarks of TSC.

ALSO FROM CSC

DYNAMITE +
"THE CODE BUSTER"

now available for UniFLEX
OS-9 version soon

DYNAMITE + is a new version of DYNAMITE, our popular 6809/6800 disassembler package for 6809 FLEX. Present users of DYNAMITE can upgrade to DYNAMITE + by sending us the original DYNAMITE diskette and $40 (plus $5 for foreign postage). DYNAMITE + does everything DYNAMITE does, and more! A cross-reference generator has been added, label files are now maintained only in text form (LABEL EQU $xxxx), and boundary file specifications have been tremendously simplified, which makes it easier to disassemble large programs containing lots of big tables.

The UniFLEX version of DYNAMITE + does everything the FLEX version does, and also automatically handles system calls and 'info' areas.

DYNAMITE + is available for $100 per copy on FLEX (specify diskette size), and $300 on UniFLEX. Foreign orders add $5 per copy for postage.

SCREEN EDITOR/WORD PROCESSOR FOR DOS6x, FLEX, OS-9

IF YOU ARE CONSIDERING UPGRADING YOUR EDITING AND PROCESSING CAPABILITIES, HERE ARE A FEW GOOD REASONS TO CONSIDER SCREDITOR...

REASON #1 - SCREEN EDITING

Screen editing (sometimes called cursor-based editing) is the most efficient way in which to enter your thoughts into the computer and during which you are entering on the screen as it you were looking at a printed page - organization becomes a matter of observation rather than guesswork.

REASON #2 - DYNAMIC SCREEN FORMATTING

Also dynamic screen formatting, the second keyword on (some of what the completed job will look like. Within the capabilities of the display terminal itself, you are constantly looking at what you would have if your job were printed. Every change you make, every addition or deletion, is automatically incorporated in a "WYSIWYG-justified, ready-to-print form. You no longer need to guess what all those cryptic formatting commands you may have used with other software will turn your finished job into!

REASON #3 - NO DOCTORING REQUIRED

SCREDITOR III is as easy to clear you CRTRL keys to send control sequences to your printer to integrate it into your system. If you can answer a series of questions about your printer and terminal's capabilities, installation is complete.

REASON #4 - NO EXTRA "HIDDEN" PROGRAM COSTS

SCREDITOR III is complete as purchased. You get screen editing, formatting, dialog like verifying and much more, all in a single program. Add it up. Even though the competition can provide many of the features of SCREDITOR III, how much will you pay? And, more important, how many programs will you have to buy?

REASON #5 - OPERATING SYSTEM SUPPORT

No one bootlegs, SCREDITOR III supports more processor and operating system combinations than any other single editing and word processing package available. OS-9 (level one and two), Flex and SSB DOS, 6809 and 6800 versions are available. Except for those commands which are possible in a particular operating system, SCREDITOR III is identical in operation in all versions.

REASON #6 - UPGRADE SYSTEM SUPPORT COSTS

If you purchase another company's product for your FLEX-9 system and decide to run OS-9, how much will you pay for the same or very similar features (if it's available. See reasons #4,#5 with SCREDITOR III, this may be significantly more than you pay 40% of the price you paid for the first one) and you may have the same great capabilities with both operating systems!

REASON #7 - CAPABILITIES

Not only is SCREDITOR III the only SCREEN/CRT editing and word processing program available, it's also the most powerful. See if the other available programs available at any price have multi-column editing. Or allow you to add and delete, align rows and columns of numeric data. Or allow you to change margins and tabs at any time during editing (not like a typewriter, where you must have to re-justify and change the width of paragraphs and lines re-format, while you watch. Or edit and re-execute commands. Or allow repeatable keyboard input and command entry replacement with a single keystroke (macros). Or let you save all of those macros to a disk file to be re-loaded for use in later edit sessions. Or edit your macros. Or...

REASON #8 - DOCUMENTATION

SCREDITOR III has the most comprehensive documentation available for any 68xx editor/word processor. The Technical Reference Manual is currently about 160 pages long. The Training and User's Guide we are now preparing will be about another 100 pages. We're also working on our new Tutorial Cassette (two hours of instruction). We even provide a HELP command which gives you the syntax of all of SCREDITOR III's commands and at least one paragraph of description on each!

REASON #9 - PRICE

Considering its capabilities, SCREDITOR III is the least-expensive program of its type available today. Period.

REASON #9 - HAPPINESS

We want your business. Think how happy you will make us if you decide to buy SCREDITOR III!

The following information should answer most of your questions about SCREDITOR III. If you have additional questions, please call during our office hours, or, better still, order the manual (when you decide to buy SCREDITOR III, we'll apply your manual purchase against the software price).

PROCESSORS SUPPORTED - 6800, 6809

OPERATING SYSTEMS - FLEX, SSB DOS, OS-9 (level I and II)

DISPLAY REQUIREMENTS - Serial terminal with computer-addressable cursor. 4800 baud minimum data rate, 9600 or better preferred. Clear EOL and clear EOS not required but highly desirable. Insert and Delete line improve display speed. Suggested terminals include Televideo, Heath/Zenith H-19, PMTPCo. Virtually all addressable-cursor terminals can be used. ANSI terminals supported by the time you read this.

Memory-mapped displays require software to provide terminal look-alike capabilities. Sample software included with FLEX and DOS6x orders.

Minimum display size is 16x64. Maximum is 255x255. No software patching is required to adapt software to display size.

Unique configurations may be supported by calls to external user-supplied sub-routines.

KEYBOARD REQUIREMENTS - ASCII character set with control codes required. Function keys generating multiple-character sequences also supported. Control code and function key assignments user-defined and changeable at any time without patching.

KEYBOARD INTERFACE - PIA parallel, ACIA serial, or external keyboard handlers. Interrupt-driven operation (queue-buffered type-ahead) is a standard user-selected option.

PRINTER HARDWARE - Current version supports standard printers. Specialty printer capabilities controllable by transmission of control or escape sequences imbedded in text. Underline, boldface, double width stimulated on printers with backspace capability. Proportional printing available soon.

PRINTER INTERFACE - Program uses existing user-supplied software to transfer characters from SCREDITOR III to printer hardware by calls to user sub-routines.

COMMAND SET - Fourty-six commands (or more by the time you read this) with multiple options on many commands. Thirty-one single key commands for often-used operations such as cursor movements, inserts, deletes, paragraph formatting, etc...

OPERATION AND DISPLAY - Text entry by 'window-type' screen/cursor editing. Commands by single-key or line entry. Continuous display of current tabs/margins, edit modes, cursor row and column, current line number, horizontal display range.

REQUIRED MEMORY - (program memory) - Program size - approximately 30 kilobytes. Minimum memory for useful editing - 32 kilobytes.

DISK MEMORY - Configuration files (3 minimum) - 1 to 3 sectors each
HELP file - disk-resident, approximately 12 kilobytes

********** AT LAST! **********

AVAILABILITY - Please allow one to two weeks for delivery. Disks are built as orders are received to allow the most current possible revision of software to be shipped.

PRICE - SCREDITOR III manual, disk $150.00
manual only (price may be applied to software order) . . . . . $ 40.00

IMPORTANT INFO

We accept Mastercharge and VISA credit card orders, personal and business checks drawn on U.S. banks and in U.S. dollars, and COD's to the continental United States. Domestic orders shipped by United Parcel Service unless otherwise requested. Overseas orders shipped by air parcel. COD's shipped by United Parcel Service only. Open accounts to government agencies, educational institutions, established accounts and Fortune 1000's only. Virginia residents add 4% state sales tax. On orders under $100, add $5 for shipping and handling. On overseas orders add $10 for shipping.

THE BEST NEWS!

In our haste to do business, too often the temptation to compromise our faith weighs heavily. Even though we have forgiveness when we confess our sins from a God who loves us, we would like to take a moment to ask forgiveness from any of our customers whom we have slighted. Orders do get lost or mis-shipped, and mistakes do happen. If we fail to completely please any of you in either our response to your order, inquiries, or for whatever reason, let us know. We will do anything in our power to satisfy your problems. We thank all of our customers for your prayers, and we pray for each of you that you will come to know the love of God in your lives that we have found!

ALFORD & ASSOCIATES
P.O. BOX 6683
RICHMOND, VA., 23230
804-320-6722

CoCo SPEECH SYNTHESIZER

The Color Computer is one of the best things to happen to computing since the SS-50 bus! At the price, there is no better way to get into personal computing, and we are now offering a speech synthesizer for the CoCo! With most of the same features of our VS-1, the SP-1 can give your CoCo the power of unlimited speech. Unlike software approaches to speech, the SP-1 does not eat your memory, and provides the clearest, easiest to understand speech we have heard!

The SP-1 is a completely assembled and tested unit which plugs directly into the ROM-PACK slot on the side of the computer. Unlike the VS-1, however, no external speaker is required, as the SP-1 routes its audio directly through the computer and into your television monitor!

We also have a version of our VOX-EDITOR to let you build your voice messages quickly and easily, and then save them to cassette when you are done. The speech files can later be read by Basic in your application programs! The comprehensive manual we supply gives you all the information you need to make your CoCo talk!

SP-1 pack and manual $179.95
SP-1 manual only (price applied to SP-1 purchase) . . . . . $ 19.95

TRANSLATOR FOR SP1

Our SC-1/COLOR translator uses the same highly accurate algorithm as our SC-1 for the VS-1, but drives the SP-1 on the color computer. The SC-1/COLOR, however, is even easier to use! When the SC-1/COLOR translator is loaded, it becomes a part of Basic! To translate and speak English text, numeric data (even punctuations if you desire), you simply program PRINT #1 followed by your text. The translator takes it, converts it to a phoneme stream, and drives the SP-1 directly!

SC-1/COLOR cassette and manual (requires SP-1 for operation) $29.95
(manual not available separately)

SSB UTILITIES DISK

Two of the shortcomings of SSB DOS are that, one, with six-character file names and default extensions it's hard to know from a name what is in a file, and, two, with no disk information sector the only way to know what is on a disk is to list the files! Since we couldn't conveniently make the file names longer, we did the next best thing and wrote a family of programs to allow you to 'name' your disks!

Utilities #1 provides a NAME utility to build a disk information sector on each newly-formatted disk. UPDATE lets you change this sector, LOCK lets you write, delete or even LIST protect the files on the disk. LIST is a better listing utility which reads the disk access code from the information sector and will only list LOCK'ed files if you supply the access code. A number of other programs are also included on the disk.

The manual provides information about the way SSB disks are formatted and how the chained sector method work in addition to complete descriptions about each of the programs provided on the disk!

UTILITIES #1 disk and manual $29.95
(manual not available separately)

FANTASTIC PROOFREADER

We are proud to offer Pete Stark's Spell 'n Fix proofreader. We examined a number of very good pieces of spelling correction software. We compared ease of use, speed, correction capability, dictionary maintenance and expansion, disk space overhead, etc... and found that Pete Stark of StarKits has done a remarkable job on this program. Look at these features and make your own decision....

MULTIPLE DICTIONARIES - Spell 'n Fix allows you to create multiple dictionaries. In this way, if you want, you can put technical and special words in a separate dictionary. Then, when you proofread your documents, select the dictionaries which have the words you need, thus speeding up the job!

EXECUTE DOS COMMANDS - Need to list your disk to decide which dictionary or text file to proofread? Want to delete an old file? View a file? Most of your DOS commands can be executed directly from within Spell 'n Fix.

MULTIPLE CORRECTION OPTIONS - With Spell 'n Fix, you can add new words to your dictionary at any time. Or, if you want, you can create a new dictionary for words not found in the existing dictionaries. Or simply print out the new words. Or even ignore them! CORRECTION OR MARKING - With Spell 'n Fix, you can mark incorrectly-spelled words in your text for later correction, or you can tell Spell 'n Fix to correct them as they are found. The choice is yours!

COMPRESSED DICTIONARY - Spell 'n Fix uses either of two forms of dictionary, and converts either to the other at your command. The first type is a fully expanded dictionary with one word per line, allowing you to use your editor to make corrections, deletions and additions easily. The second type incorporates a slick compression method to drastically reduce dictionary size and increase access speed.

After looking at our options, we decided that this was the program for us, and are proud to be the exclusive agents for StarKits for the SSB DOS68 and DOS09 versions. We can also supply versions for 6800 and 6809 Flex. The OS-9 version will be available soon directly from StarKits. StarKits also has a version for the Color Computer! At the price, we feel that no other program can do what Spell 'n Fix can....

SPELL 'N FIX disk (with 17000 word dictionary) and manual $178.50
(manual not available separately)

8S50 SPEECH SYNTHESIZER

The VS-1 is a completely-assembled, tested unit for the SS-50 I/O bus which gives your computer the power of unlimited speech. Since the VS-1 is programmed phonetically, there is essentially no limitation on what you can make your computer say! The VS-1 programs just like a parallel printer driver, and requires an average of only one phoneme per letter in English for a very low message memory storage overhead. The data rate is typically a low 70 baud.

The VS-1 is supplied with the easiest-to-use, most flexible speech editing program available. This program makes building your speech files or data a real pleasure. We also supply additional software and sample speech files on the disk to give you a 'leg up' on making your computer talk.

The manual provides sections on speech generation theory, operation of the supplied software, complete technical descriptions, a sample speech dictionary and phonetic code charts, fifty pages in all!

If you are a roll-your-own nut, we provide the bare board or manual or disk or synthesizer chip, or combinations (We suggest that only VERY experienced persons attempt such assembly due to the fact that we do not provide assembly instructions beyond parts list parts layout and schematic).

The VS-1 is available for 6800 and 6809 FLEX, 6800 and 6809 SSB DOS, and OS-9 level I and II.

VS-1 complete with manual and disk $194.95
VS-1 manual and disk only (may be adapted for brand-X) . . . . . $ 29.95
VS-1 manual only (may be applied to VS-1 order. Look before you leap!) . . . . . $ 19.95
VS-1 bare board (build your own...requires experience!) . . . . . $ 19.95
SC-01A synthesizer chip . . . . . $ 40.00
VS-1 bare board, chip, disk and manual . . . . . $ 99.95

TRANSLATOR FOR VS1

If you have our VS-1 (or even brand-X), and want an easier way to make your computer talk, our SC-1 translator is for you. The SC-1 takes plain English text from a disk file or your keyboard, translates it to a phoneme stream, and sends the resulting output to either the speech synthesizer hardware or to a data file for later use.

The SC-1 translator is one of the most accurate such programs we have seen, making typically only one 'mistake' per sentence, for an average accuracy of about 95%! Due to the type of algorithm we are using, even proper names are generally translated correctly! Where the SC-1 is being used to compile 'canned' messages for later use, we provide an additional program which takes a binary phoneme file created by the SC-1 and converts it to a speech file which may be edited by our VOX-EDITOR (supplied with the VS-1). In this way, you can quickly create your messages and then 'fine-tune' them!

The SC-1 translator is available for 6800 and 6809 FLEX, 6800 and 6809 SSB DOS, and OS-9 level I and II. If you are using brand-X, order the SC-1 with the VOX-EDITOR on the same disk!

SC-1 disk and instructions $39.95
SC-1 disk, instructions with VOX-EDITOR $49.95

TREK-68

Our TREK program is the best 'real-time', 'live-action' game available for 68xx computers using memory-mapped displays. This program does not require graphics capabilities from your display. Otherwise, this is a full arcade-type game. No trivitality here, either. The suicide option has only been won by a handful of people world-wide, and can run for hours!

TREK is available for 6800 and 6809 FLEX and 6800 and 6809 SSB DOS, and requires both a memory-mapped display and a source of 60 or 100 Hz interrupts (such as the MP-T timer board).

TREK disk and manual $24.95

COLOR-TREK

Like our other version of TREK, COLOR-TREK provides you seat-of-the-pants action. This program uses no routines from Basic, and so can run on non-extended Basic machines, 16k memory is required. Only one player to our knowledge (other than the author) has won the suicide option of this game, and it took him fifty-seven minutes! No trivitality here!

COLOR-TREK cassette and manual $17.95

Universal Data Research, Inc. Introduces

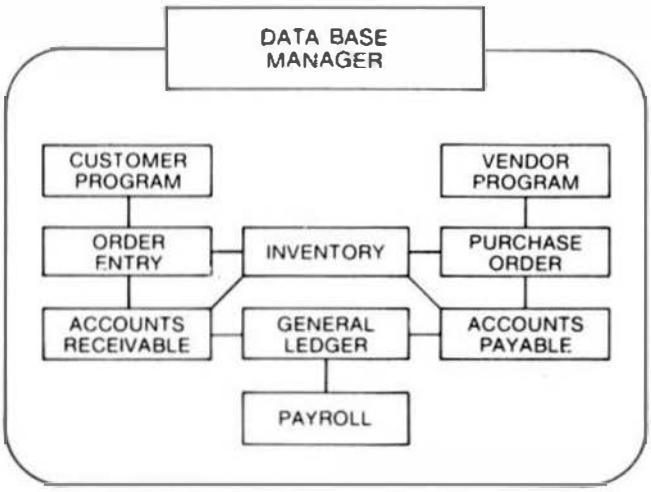
MODEMS

300 Baud Acoustic	$149.00
300 Baud Direct	$179.00
300 Baud Auto Answer	$219.00
1200 Baud Direct	$449.00
1200 Baud Auto Answer	$499.00
300/1200 Auto Answer	$549.00
300/1200 Auto Dial	$599.00
Modem 2 Port I/O Card	$119.00
with speed select	
Printer Special: Okidata 82A	$449

Dealer Inquiries Welcome ● Call or Write For Free Catalogue

New, Lower Prices

FLEX* and UniFLEX*
software for the 6809



Integrated Business Programs

	FLEX	UniFLEX
Accounts Payable	$295	$395
Accounts Receivable	$295	$395
General Ledger	$295	$395
Inventory 2	$295	$395
Payroll	$295	$395
Data Base Manager	$350	$450
Word Processing		
Software	$295	$395
WP Menu		$150
P Control		$150

UNIVERSAL DATA RESEARCH INC.

*FLEX & UniFLEX are Trademarks of Technical Systems

2457 Wehrle Drive, D-68, Buffalo, NY 14221
PHONE (716) 631-3011

Dealer Inquires Welcome

master charge VISA

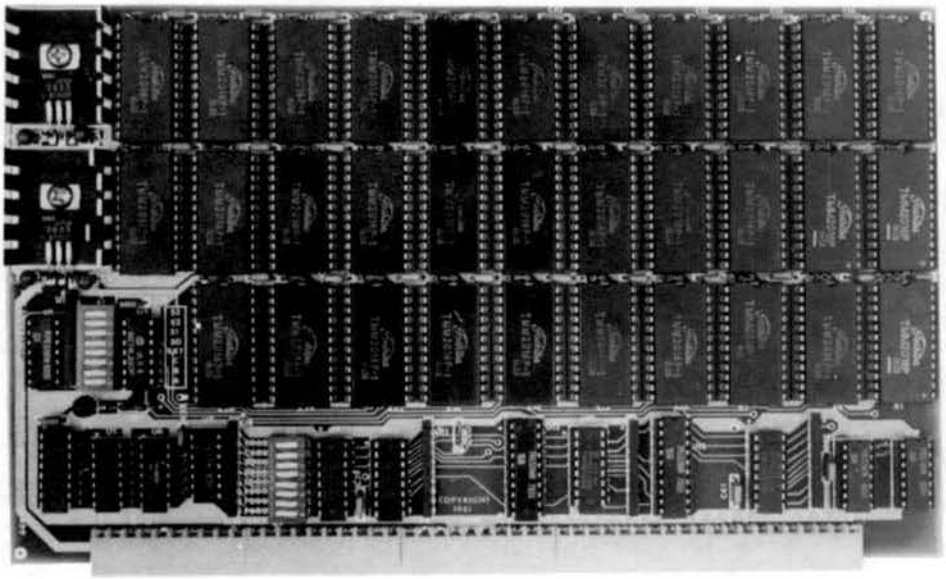
64K SS-50 STATIC RAM

NEW!

$199.00 (48K KIT)

NEW!

LOW POWER!



RAM OR EPROM!

BLANK PC BOARD WITH DOCUMENTATION $52

SUPPORT ICs + CAPS -	$18.00
FULL SOCKET SET -	$15.00

ASSEMBLED AND TESTED **ADD $40**

FEATURES:

- ★ Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
- ★ Fully supports Extended Addressing.
- ★ 64K draws only approximately 500 MA.
- ★ 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
- ★ Board is configured as 3-16K blocks and 8-2K blocks (within any 64K block) for maximum flexibility.
- ★ 2716 EPROMs may be installed anywhere on Board.
- ★ Top 16K may be disabled in 2K blocks to avoid any I/O conflicts.
- ★ One Board supports both RAM and EPROM.
- ★ RAM supports 2MHZ operation at no extra charge!
- ★ Board may be partially populated in 16K increments.

56K	$249
64K	$299

16K STATIC RAMS?

The new 2K x 8, 24 PIN, static RAMs are the next generation of high density, high speed, low power, RAMs. Pioneered by such companies as HITACHI and TOSHIBA, and soon to be second sourced by most major U.S. manufacturers, these ultra low power parts, feature 2716 compatible pin out. Thus fully interchangeable ROM/RAM boards are at last a reality, and you get BLINDING speed and LOW power thrown in for virtually nothing.

Digital Research Computers
(OF TEXAS)
P.O. BOX 401565 • GARLAND, TEXAS 75040 • (214) 271-3538

TERMS: Add $2.00 postage. We pay balance. Order under $15 add 75¢ handling. No C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50, add 85¢ for insurance.

ARCADE 50

POWERFUL COLOR GRAPHICS
Uses the new TMS9918A Video Display Processor. High resolution 256 x 192 pixel display with 15 colors.16k Bytes of onboard RAM does not reduce user memory. 32 graphic images can be individually moved with simple X-Y commands for smooth animation.
External Video input allows subtitling.
NTSC composite video output
SOUND EFFECTS AND MUSIC
Three AY3-8910 Programmable Sound Generators
Nine simultaneous voices
Three independent noise sources
Onboard stereo amplifier drives two 8 ohm speakers
ADDITIONAL I/O CAPABILITIES
Eight analog inputs with 8 bit resolution
Supports four joysticks with pushbutton switches
Eight bit parallel I/O port
Entire unit maps into 256 bytes of memory
DOCUMENTATION AND SOFTWARE
Programming manuals for Video and Sound Processors
Subroutine library and Super Demo Maze Game
Example programs in BASIC, FBASIC and ASSEMBLY
User library and sales support

ARCADE 50 assembled and tested	$325.00
Video and Audio connector set	15.00
4 Joystick connector set	15.00
2 Radio Shack Joysticks	24.00
UHF channel 33 modulator	32.00
Gold Molex connectors	12.00
A/BASIC for 6800	110.00
FBASIC for 6809	110.00
FBASIC (with ARCADE 50)	75.00
FBASIC (manual only)	10.00
ARCADE 80 (TRS Model I)	395.00
ARCADE 100 (S-100 BUSS)	375.00
ARCADE 50 RGB	375.00
LABVIDEO (Motorola EXORbus)	375.00
LABVIDEO RGB	375.00
NEW MV096809 Processor Board	225.00
*Comes assembled with PIA and ACIA	
*12 Sockets for 2716, 2732 or RAM	
*Supports DMA disk I/O	
*Ideal for 6809 upgrade or process control	
AMDEK COLOR I Monitor	425.00
AMDEK COLOR II Monitor	799.00
AMDEK COLOR III Monitor	499.00
256K Dynamic Memory Board (assembled)	795.00
256K Dynamic Memory Board (assembled w/64K)	395.00
64K Dynamic Memory Board (assembled)	295.00

Specify 5" or 8" soft sector disk for TSC's FLEX or MICROWARE'S OS/9 system.
TERMS: CASH, VISA, MC, C.O.D.

FBASIC

TERMINUS DESIGN INC., in conjunction with Microware Systems Corporation, is proud to announce FBASIC-an enhancement of Microware's 6800 A/BASIC. Their fast compiled BASIC has been adapted for 6809 users with added video and sound features for ARCADE 50 users. FBASIC is a true compiler that produces optimized machine language modules which are ROMable and require no Run-Time package. FBASIC requires less memory overhead and runs hundreds of times faster than BASIC interpreters. It supports standard BASIC instruction including String functions. Disk I/O and fast integer arithmetic with multiple-precision capability. Graphics verbs and functions fully support the Arcade 50. Arcade statements include:

INIT	MODE	BLANK	BACKDROP
SIZE	MAG	VREG	DELAY
MOVE	DRAW	FCOLOR	JSWITCH
REMOVE	RDRAW	BCOLOR	SWITCH
PSG	TONE	ENVL	VOLUME
ADC	SPRITE	SPNAME	ENDEF
SPCOLOR	RSPRITE	SPDEF	PATDEF
VPEEK	VPOKE	VPRINT	

TERMINUS DESIGN INC
16 SCARBROUGH ROAD
ELLENWOOD, GA 30049
(404) 474-4868

68 MICRO JOURNAL PROGRAMS on DISK

Disk #1: FILESORT, MINICAT, MINICOPY, MINIFMS, **LIFETIME, *POETRY, **FOODLIST, **DIET.
Disk #2: DISKEDIT w/ inst. & fixes, PRIME, *PRMOD, **SNOOPY, **FOOTBALL, **EXPAWN, **LIFETIME.
Disk #3: CBUG09, SEC1, SEC2, FIND, TABLE2, INTEXT, DISK-EXP, *DISKSAVE.
Disk #4: MAILING PROGRAM, *FINDDAT, *CHANGE, *TESTDISK.
Disk #5: *DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk #6: **PURCHASE ORDER, INDEX (Disk file indx).
Disk #7: Linking Loader & RLOAD, Harkness
Disk #8: CRTSET, Lanpher (May '82)
Disk #9: DATECOPY, DISKFIX9 (Aug '82)

NOTE: All are as published or received by 68 Micro Journal, some have fixes and patches.

This is a reader service only! No Warranty is offered or implied, they are as received and are for reader convenience ONLY. Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other.

PRICE: 8" Disk $19.95 - 5" Disk $17.95

68 MICRO JOURNAL
POB 794
Hixson, TN 37343
615-842-4600

* indicates 6800, ** indicates BASIC SWTPC or TSC - 6809 no indicator.

MASTER CARD - VISA accepted - Foreign add sufficient postage surface or air!!

WINCHESTER FOR MOTOROLA EXORCISOR/MDOS

☐ 10 MB Winchester hard disk runs MDOS on Motorola Exorcisor System ☐ No modification to MDOS required ☐ MDOS based software stays alive ☐ All user software operates without modification ☐ Optional SA-801R flexible diskette drive system ☐ Optional 10 MB removable cartridge.

CSA *For information call (619) 566-3911*
Computer System Associates
7562 Trade Street, San Diego, CA 92121

AAA Chicago Computer Center

ELEKTRA CABIN † Made of heavyweight 0.090" thick aluminum. Interior is 18-1/2" wide by 21-7/8" deep by 6-3/4" high. Heavy duty A.C. line cord, A.C. fuse holder, EMI filter. Fan with filter. Back panel has 10 cutouts for 'D' type data connectors. Front panel has key on/off power switch, 2 illuminated push button switch (Reset and NMI/Abort), and two cutouts for 5-1/4" disk drives.

POWER SUPPLY Highest quality linear power supply CONSERVATIVELY rated at 15a @ 8v, 3a @ 16v, 3a @ -16v. 3 primary inputs for light, rated, and heavy loading.

DISK REGULATOR BOARD WITH CABLES Standard version for 2 floppy drives. Heavy duty version for 1 Winchester drive and 1 floppy drive.

ELEKTRA UNIVERSAL MOTHERBOARD Heavyweight 0.125" thick, 18" long by 9" wide. 11 memory (50 pin) slots. 4 or 8 slots may be cut off for shortening to 14" or 10" lengths respectively. 8 I/O (30 pin) slots. Complete address decoding of I/O slots. Choice of 4, 8, or 16 addresses per I/O slot. Base address for I/O slots can be placed at 32, 64, or 128 byte increments respectively. 1" spacing between all memory and I/O slots. Extended addressing capability for both memory and I/O ports for meeting SS-50C bus specifications. On board baud rate generator with low and high ranges providing jumper selectable rates of 75 through 38,400 for each of the five baud rate lines. Slow device circuitry permitting 1 MHz 30 pin disk controllers to run with 2 MHz 50 pin CPU boards.

ELECTRA CPU 8/9 Use either the 6802 or 6808 (to run 6800 software) or 6809. Has provision for up to 3 2716 Eproms, 1K scratchpad, MC6840 triple timer, and an optional baud rate generator providing baud rates from 110 through 38,400 baud in two user selectable ranges. The board supports DMA by either HALT or BUSREQ when a 6809 CPU is used. The board does not support a DAT and therefore does not support extended addressing. The board will run any of the MIKBUG™ compatible monitors in the 6802/6808 mode and SBUG-E, HUMBUG, and MICROBUG in the 6809 mode. The ELEKTRA CPU 8/9 will run any of the popular disk controller boards with the appropriate software. OS-9™ Level 1 is available when used with the SSB DCB-4A controller or Gimix compatible single density controllers.

ELEKTRA DPS DUAL PORT SERIAL CARD Fits the standard 30 pin SS-50 bus I/O slot. Can be configured for 4 addresses per port with the B port 2 addresses higher than the A port or for 16 addresses per port with the B port 4 addresses higher than the A port. Each port is terminated at two 16 pin dip sockets, one socket configured for modem and the other for terminal or printer. RTS, CTS, DTR, DCD, and DTR are appropriately implemented. Each port has independent selection of baud rate. Each port allows the interrupt request to be independantly jumpered to the IRQ or FIRQ/NMI bus line.

ELEKTRA DPP DUAL PORT PARALLEL CARD Fits the standard 30 pin SS-50 bus I/O slot. Can be used in either the 4 or 16 addresses per I/O slot configuration occupying the first four addresses of the I/O slot. The direction of the TTL buffers can be controlled by either on board jumper connectors or by a signal from the peripherals. The interrupt request lines for each port may be individually jumpered to either the IRQ or FIRQ/NMI bus line.

ELEKTRA CHASSIS Includes cabinet, 110v power supply, power supply cables, standard disk regulator board with power cables, assembled motherboard with gold square pin connectors $825.00*

ELEKTRA CABINET	260.00
ELEKTRA 110v POWER SUPPLY	175.00*
ELEKTRA CABINET, 110v POWER SUPPLY, AND POWER SUPPLY CABLES	410.00*
ELEKTRA CABINET, 110v PS, PS CABLES, STD. DISK REGULATOR AND CABLES	460.00*
*Add $30.00 for 220v	
STANDARD DISK REGULATOR BOARD AND CABLES	50.00
HEAVY DUTY DISK REGULATOR BOARD AND CABLES	75.00
FILLER PLATE FOR 5-1/4" DRIVE OPENING	10.00
FAN FILTER	10.00
ELEKTRA D-5 Dual drive cabinet for 5-1/4" drives with power supply, line cord, fuse, power switch, and power cable to drives	125.00
ELEKTRA MD-5 (Heavy duty version of D-5 package above)	150.00
ELEKTRA SHD-5 (Super heavy duty: Powers 1 Winchester and 1 floppy)	175.00
5' ribbon cable for dual 5 1/4" disk drives	40.00
ELEKTRA D-8 Dual drive cabinet, power supply, ps cable for 8" drives	350.00
Cabinet for dual 8" drives only	250.00
Power supply for dual 8" drives only	120.00
PS cables only (Specify brand and type of 8" drives)	30.00
6' ribbon cable for dual 8" disk drives	45.00
GOLD 10 PIN CONNECTORS (Specify male w/square pins or female)	1.50
TIN 10 PIN CONNECTORS (Specify male w/square pins or female)	0.50
TITANIUM-TIN 10 PIN FEMALE CONNECTORS (While supply lasts)	0.50

	BARE BOARD	KIT	ASSEMBLED
ELEKTRA UNIVERSAL MOTHERBOARD (Gold)	80.00	320.00	380.00
ELEKTRA UNIVERSAL MOTHERBOARD (Tin)		240.00	300.00
ELEKTRA CPU 8/9 (Add $20.00 for MICROBUG)	50.00	225.00	275.00
ELEKTRA CPU 8/9 baud rate option		25.00	25.00
ELEKTRA OPP DUAL PORT PARALLEL BOARD**	20.00	60.00	80.00
ELEKTRA OPS DUAL PORT SERIAL BOARD**	20.00	60.00	80.00
**CABLE FOR DPP OR DPS (2 needed, specify board)			20.00

WARNING AAA Chicago Computer Center does not provide repair or diagnostic service for customer assembled kits. AAA Chicago Computer Center does warranty and maintain service for our assembled boards. The customer should carefully take into consideration the small differential seperating our kit and assembled prices when making his choice of purchase.

We have introduced our line of computer equipment with the purpose of offering the highest quality of components possible at affordable prices. These products are intended for OEM applications where it is the responsibility of the purchaser to integrate these components with suitable memory, disk controllers, drives, and software along with I/O terminals to form working computer systems.

AAA Chicago Computer Center

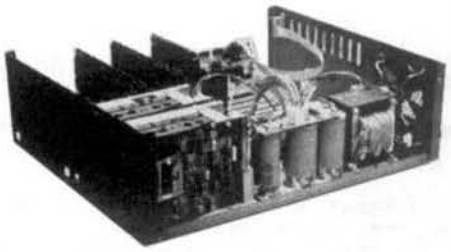
120 CHESTNUT LANE • WHEELING, IL 60090
(312) 459-0450

Technical consultation available 4 PM to 6 PM most weekdays. Closed evenings and weekends.

TERMS Minimum order $20.00 Shipping and handling estimates within the Continental U.S., add 3% (MINIMUM $2.50). Illinois residents add 6% sales tax. We will refund your overestimated shipping and handling charges. Foreign shipping and handling, add 10% (MINIMUM $10.00). Foreign orders must be prepaid in U.S. dollars. Heavy foreign items will be shipped air freight collect. Please phone between 4 PM and 6 PM weekdays if questions arise regarding shipping fees. Master Charge, Visa, and American Express honored.

Our apology. We are not staffed to answer technical inquiries through the mail. Please phone for technical help during the hours indicated above. The too frequent changing of our inventory and prices makes it uneconomical to publish a catalog. Our ads are intended to serve that purpose. Prices and inventory are subject to change without advance notice.

ELEKTRA
COMPUTER PRODUCTS



SMOOTH™ Software

(All of our software is copyrighted and all rights are reserved. Source is either supplied or optionally available at extra cost so that the purchaser can modify our programs for his own use. Licensing, however, is required for commercial resale.)

SUPER MODEM PROGRAM

Hardware independent (No interrupts required). Assumes modem is connected to an MC6850 (serial interface) and the control port is connected to either an MC6850 or MC6820 when used with the Video GMXBUG.

Transmit manually to distant computer

Transmit disk files (text) of any length to distant computer

Receive and save disk files (text) of any length on local disk system. If sending computer does not support an X-on/X-off protocol, then the received files are limited in size by the computer memory.

Tested to transmit and receive text at speeds up to 9600 baud. (CRT terminal must be capable of operating at a baud rate higher than the one the modem is operated at.)

Half duplex option in case distant computer doesn't echo

Echo option so user can simulate a time sharing system. (Super Modem Program doesn't support auto-answer but the source is provided for those individuals who wish to adapt our program to their special needs.)

Replaces CR with CR/LF (user option) for those using time sharing systems that don't transmit LF's

Slow disk file transmit based on character verification (plus user installed timing loops if necessary) for use on time sharing systems to which disk files cannot be sent at speeds suggested by the baud rate.

Please specify 6800 or 6809, SSB or FLEX™, 5" or 8".
Manual and disk with both source and object code $75.00

STANDARD MODEM PROGRAM

Same as Super Modem Program above but without ECHO option, CR/LF for CR option, slow disk file transmit option, nor X-on/X-off option. Reception of disk files is limited to those small enough to completely fit within the receiving buffer.

Please specify 6800 or 6809, SSB or FLEX™, 5" or 8".
Manual with instructions, source listing, and flow chart; disk with both source and object code $45.00
Manual with instructions, source listing, and flow chart 25.00

MODEMS (By U.S. Robotics) Auto-Dial is Hayes compatible	MANUAL ANSWER	AUTO ANSWER	AUTO-ANS AUTO-DIAL
300/1200 Baud, direct connect	N/A	499.00	549.00
1200 Baud (120 cps), direct connect	399.00	449.00	N/A
300 Baud (30 cps), direct connect	159.00	199.00	N/A
300 Baud (30 cps), acoustic	129.00	N/A	N/A

SMOOTH™ Software

ALL IN ONE
Editor + Text Processor + Mailing Labels
Mailing Lists + Use any CRT terminal and printer

Supports Editing commands such as bottom, change, delete, find, insert (single line), input (multiple lines), list, next, overlay (with cursor editing, character deletion and insertion), overstrike (for selected dark or text), print, restart-set-top, underline, up, and verify.

Supports Text Processing commands such as block copy, block move, centering, margin justification (widen and narrow), paging, and tabbing.

Mailing Lists and Labels. Use the same mailing list disk file (with Projected areas) for both mailing labels and repeat letters. Repeat letters are personally adressed to each person or selected persons on the mailing list.

Most Powerful File Handler found in any editor. Append one file to the end of another, or insert (merge) one file into another as designated by the line pointer. Print specified lines to your printer or to a disk file. Edit files larger than the text buffer. Does not produce output files when not desired. Delete disk files from the editor.

Printer commands. Control characters can be sent to the printer for formatting control either directly from the control terminal or by imbedding them in the text. The set command contains interface initialization and character output routines to support the SWTPC MP-C interface as well as the standard serial and parallel interfaces. JumPs are also provided to user supplied printer routines. User selects the port address (0 thru 7, A or B) thereby eliminating the need for the user to install printer software routines. Editor can be initialized for either 4 or 16 addresses per port.

Editor allows exiting to either the monitor or DOS and then reenter (Warm Start) without destroying previously prepared text in the buffer. The Restart command erases contents in the buffer without the user having to reload the Editor.

The Editor allows the user to toggle between full duplex (no echo) and half duplex (echo) as needed. It responds to commands in both upper and lower case and can be used to create assembler source code and Basic programs as well as text.

Specify 6800 or 6809, SS8 or FLEX™, 5" or 8"	75.00
Printed source listing is available for an additional	35.00
All-In-One, Write'n Spell, and Spell'n Fix package	195.00

Software by Technical Systems Consultants, Inc.	UniFLEX™ w/1 yr. mnt.	FLEX™
DOS (includes Editor and Assembler)	550.00	150.00
Editor		50.00
Assembler		50.00
68000 Cross Assembler on 6809	300.00	250.00
6809 Cross Assembler on 6800		100.00
Text Processor	150.00	75.00
Extended Basic	200.00	100.00
Basic Precompiler (specify standard or extended)	150.00	50.00
Pascal(Flex™)	300.00	200.00
Sort/Merge Package	150.00	75.00
6809 Flex™ Utilities	150.00	75.00
6800 Flex™ Utilities		100.00
Debug Package	75.00	75.00
Diagnostic Package	75.00	75.00
6809 Relocating Assembler & Linking Loader	175.00	150.00
Fortran (Requires Relocating Assembler & Linking Loader)	350.00	275.00
Fortran (With Relocating Assembler & Linking Loader)	450.00	375.00
Cobol	750.00	500.00

Software by Microware Systems Corp.	Run-Time Package	Update	Source	Manual	Object
OS-9™ Level One Operating System		75.00	400.00	40.00	200.00
OS-9™ Level Two Operating System		75.00	N/A	40.00	500.00
BASIC09™	100.00	75.00	N/A	25.00	200.00
OS-9™ Macro Text Editor			300.00	15.00	125.00
OS-9™ Interactive Assembler			300.00	10.00	125.00
OS-9™ Interactive Debugger (Disk version)			100.00	10.00	50.00
CIS Cobol Compiler	400.00	50.00	N/A	60.00	300.00
Pascal Compiler	100.00	100.00	N/A	40.00	400.00
Microware yearly support service ($200.00 for OS-9 Level 2)					75.00

SWTPC	KIT	ASSEMBLED
6809 SWTPC FLEX™ Disk and manual (Disk only 15.00)		35.00
DC-4 Disk Controller (SS/OS, SO/DD, 5-1/4")	N/A	230.00
SBUG-E (2716 compatible)		25.00
MP-S2 Dual port serial or MP-L2 dual port parallel	N/A	120.00
MP-R Single voltage 2716 prom programmer	N/A	114.50
MP-N Calculator board	54.95	92.00
MP-T Interrupt timer	N/A	92.00
S32 Universal Static Memory Board	N/A	124.50
MP-09 6809 CPU board (2MHz)	N/A	295.00

Special Software

MICROBUG 12K, 6809 Baby HUMBUG by Peter Stark	30.00
4K 6809 HUMBUG	75.00
4K 6800 HUMBUG (RAM needed at $A000 and $D000)	65.00
2K 6800 HUMBUG (With cassette LOAD and PUNCH)	40.00
2K 6800 HUMBUG (Extra commands instead of cassette software)	40.00
Other HUMBUG versions including video versions are available	
Spell'n Fix by Peter Stark	89.29
Write'n Spell by Peter Stark	75.11
All-In-One, Spell'n Fix, and Write'n Spell package	195.00
Dynamite Disassembler	60.00
Dynamite + (Update. Send original diskette plus $40.00)	100.00
SUPER SLEUTH Disassembler System ($101.00 for OS-9 version)	99.00

SMOOTH and ELEKTRA are trademarks of AAA Chicago Computer Center
FLEX and UniFLEX are trademarks of Technical Systems Consultants, Inc.
OS-9 and BASIC09 are trademarks of Microware Systems Corp.
GIMIX and GHOST are registered trademarks of GIMIX Inc.
CP/M is a registered trademark of Digital Research Inc.
Z80A is a registered trademark of Zilog, Inc.

Dealer for GIMIX, SSB, SWTPC, Microware Systems Corp. and Technical Systems Consultants, Inc.

Prices and inventory are subject to change without advance notice. This ad is our catalog.

See our ad on the previous page to your left for ordering instructions.

SPECIALS

- New Televideo Z80A, 4MHz, 64K computer system with 950 compatible built-in terminal, dual 5-1/4" floppy drives (1 Mbyte capacity total), CP/M — 3295.00
- Same but with one floppy and one 10Mbyte Winchester — 5495.00
- Six user versions available — Call
- While Supply lasts, titanium-tin 10 pin female connectors — 0.50
- SWTPC MP-09B, socketed, with baud rate generator (only 1 left) — 250.00
- SWTPC MP-A2, socketed, all with crystal clock 1 MHz (only 4 left) — 115.00
- SSB BFD Controllers (only 5 left) — 299.00
- GIMIX 32K Memory Boards with 32K RAM (only 4 left) — 199.00
- GIMIX #76 Video Board (only 1 at this price) — 399.00
- Remex 8" DS/DD Disk Drives (only 4 left) — 399.00
- MICROBUG (2K, 6809, Baby HUMBUG by Peter Start) — 30.00
- Cat 300 Baud Accoustic Modem (only 1 left) — 99.00
- GIMIX 6800 CPU w/baud rate and timer, used (only 1 left) — 199.00

DISK DRIVES 30 day guarantee	1 head flippy SD DD Capability MPI	Siemens	2 heads MPI	2 heads QUME
5-1/4", 40 tracks	250.00	260.00	325.00	350.00
5-1/4", 80 tracks	325.00	335.00	425.00	450.00
Service Manual (Specify 40 or 80 track)			20.00	35.00
8", 77 tracks SD/DD Qume (Remex $399.00)				475.00
Service Manual (Qume DT-8)				40.00

SPECIAL BOARDS

Microtime II Calendar and Clock Board	80.00
Data Mart 16K EPROM bareboard (2708 chips)	30.00
Data Systems "68" Extender Board (Specify 30 pin or 50 pin)	25.00

TERMINALS

Hazeltine 1420	399.00
Hazeltine Espirit I (Espirit II $549.00)	499.00
Adds Viewpoint Green Screen	525.00
Televideo 925 Green Screen	749.00
Televideo 950 Green Screen (Std. Keys $895.00)	949.00
Televideo 950 Black Screen	795.00

Printers

Okidata ML 82A (120 cps, 9x9, bidirectional, serial and parallel)	475.00
Tractor for ML 82A	80.00
Okidata ML 83A (120 cps with tractor)	775.00
2K byte buffer (High speed RS-232)	125.00
Dot addressable graphics	50.00
Okidata ML 84, 200 cps, 2K, Graphics, Parallel (Serial $1295.00)	1195.00
TI 810 w/lower case and full vertical forms control (limited quantity)	1400.00
Florida Data (800 cps)	3495.00
NEC 3510 Std.	1495.00
NEC 7710 Std.	2445.00
Spare ribbon cartridge for Epson MX-80(FT) (While supply last.)	7.99

Optimal Technology, Inc. EP-2A-79 Eprom Programmer (Personality Modules extra for above programmer) — 169.00

Optimal Technology, Inc. 30 pin parallel I/O board for EP-2A-79	37.00
Software package for EP-2A-79 (Specify 6800 or 6809)	30.00
Windrush Eprom Programmer for SS-30 (Add $20.00 for Exorcisor)	375.00

Smoke Signal Broadcasting

DCB-4A Double Density Controller Board for 5" and 8" with DOS	349.00
SSB DOS (Specify 6800 or 6809, BFD or DCB-4A, 5" or 8")	75.00
SE92/SA92-5 (6809 Edit/Assm for DOS)	89.95
SSB Monitor (Specify 6800/6809, $8008/$E008/$F7E8)	75.00
SS8 version of FLEX™ (without Editor and Assembler)	150.00
LMB-1A Motherboard	399.00
SCB-69 6809 CPU Board	399.00
PAR-1 Dual Port Parallel Board	89.00
SER-2 Dual Port Serial Board with 2 Cables	129.00
Chief 90 64K Computer System	2195.00
Chief 9524 64K Computer System with DS-DT-DD 5" FD	4325.00
Static Memory Boards M-16-X 195.00 M-24-X 295.00 M-32-X 395.00	
Dynamic Memory Boards M-128-X 995.00 M-256-X 1295.00 M-512-X 1895.00	

GIMIX

6800 CPU Board(Timers $64.03, Baud rate $30.00, 2 MHz $15.00)	224.03
2 MHz 6809 Plus CPU, time of day clock, battery backup, 1K NMOS RAM	578.05
GIMIX Dynamic Address Translator (SWTPC Compatible DAT $15.00)	35.00
Filler plate for 5-1/4" drive opening	14.92
Baud rate generator board	88.93
Missing cycle detect card	38.23
50 Pin Prototyping board (30 Pin Prototyping board $36.33)	56.66
5" Single Density Controller (Without 1771 chip $158.38)	198.48
5" and 8" Single Density Controller	226.58
5" Double Density Controller	298.28
DMA 5" AND 8" Double Density Controller	588.68
GIMIX version of FLEX™ (without Editor and Assembler)	90.00
Double disk regulator card	68.22
Ribbon Cable, Two 5 1/4" inboard drives $34.98, Two 8" outboard drives	44.26
8" disk ribbon cable and back panel connector set	29.25
8" disk drive cabinet with power supply	848.18
19MB Winchester and controller update (38MB $6888.91)	4288.90

Memory	CMOS WITH BAT. BACKUP	NMOS NO BAT. BACKUP
64K Static RAM Board with 24K of RAM installed	N/A	348.27
64K Static RAM Board with 32K of RAM installed	518.36	398.37
64K Static RAM Board with 48K of RAM installed	N/A	518.47
64K Static RAM Board with 56K of RAM installed	728.56	578.57
64K Static RAM Board with 64K of RAM installed	799.64	638.67
16 Socket EPROM/ROM/RAM Board		236.32
8K Promboard (2708)		98.34

I/O Boards	1 port	2 port	8 port
Serial interface	88.41	128.43	318.46
Parallel interface		88.42	198.45
Cable sets for above boards (specify board)		each	24.95

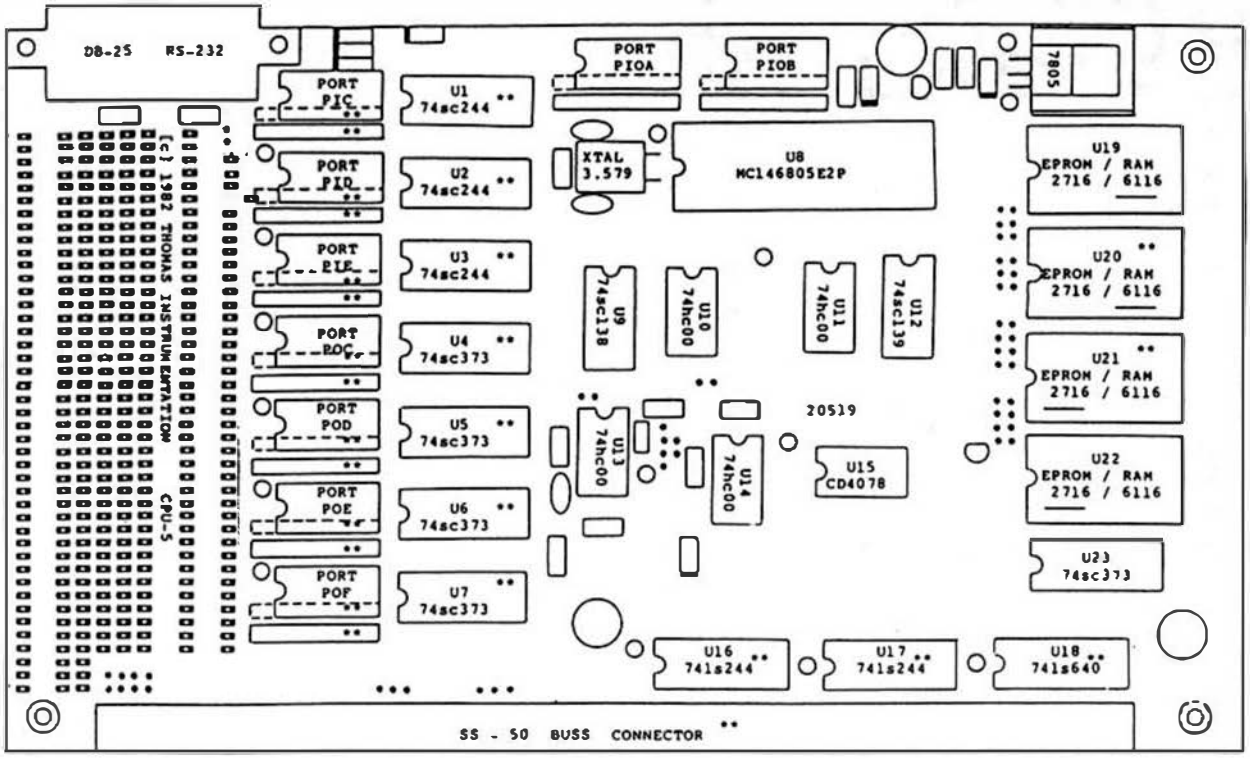
2MHz 6809 PLUS Computer System w/o Disk Cont.	w/#58 Cont.	w/#68 Cont.
With 56K of memory 2498.29	2988.50*	3248.49*
With 128K of memory		3788.39
With 128K and 19MB Winchester		6998.09
With 128K, 38MB Winch. dual 8"		17488.88

*Includes GMXBUG/FLEX/OS-9 software selectable
*With CMOS RAM and Battery Backup — add 150.00

Mainframe (Chassis, PS, Switches, Fan, MotherBoard, Baud Rate Gen.) 1199.19

AAA Chicago Computer Center

*** NEW 6805 ALL CMOS CARD ***



CPU-5-B	Blank card with documentation	$49.00
CPU-5-P	Partial card (parts marked ** not incl.)	$229.00 } Includes EPROM Monitor
CPU-5-F	Full card with all parts shown (fully assembled)	$349.00 } and 115 vac Power Pak.
CROSS-ASSEMBLERS	for FLEX ((c) TSC) 6809 system	$150.00

SOFTWARE

VDISK Treat extended memory as a super fast disk drive

6809 source & object	$149.00
6809 object	$ 99.00

OUTSIDE MODEM PROGRAM incl. source

UniFLEX version	$100.00
FLEX version 6800 & 6809	$ 50.00
HAYES SMART MODEM	$249.00

CROSS ASSEMBLERS for 6800,6801,6805 runs on 6809 FLEX $150.00

TV-EDIT Screen oriented Editor for FLEX 6809 $ 95.00
Please specify 5" or 8" Disk when placing order.

CHIPS

Please note when ordering IC's that a $100.00 Minimum is in effect Orders less than the minimum will be charged an additional $10.00 for handling

6116P-3 2MHZ	$13.50
2016P-2 2MHZ	$10.50
2716 1MHZ	$ 6.50
2716-1 2MHZ	$16.50
146805E2P	$25.00
68B02	$10.00
68B09	$25.00
68B21	$ 6.50
68B40	$12.00
68B50	$ 6.75
74LS640	$ 3.25

HARDWARE

NEW MEMORY PRICES TO INTRODUCE VDISK

S-R/R without memory chips	$120.00
with 8K NMOS@2MHZ	$169.00
with 16K NMOS@2MHZ	$199.00
with 32K NMOS@2MHZ	$299.00
with 48K NMOS@2MHZ	$399.00
Bare Card	$ 49.00

Extender Cards assemb. with logic aid

SS-50/50C	$ 35.00
SS-30	$ 25.00
SP-1 Prototype Board A/T	$195.00
SP-1 Bare Card	$ 49.00
SS-30 Wire Wrap Board B/C	$ 39.00
SS-30 Wire Wrap Board B/C	$ 20.00
6802 Super CPU A/T	$235.00
6802 Super CPU B/C	$ 59.00
Video RAM, B/C $49.00, A/T	$195.00
Parallel I/O, B/C $49.00, A/T	$139.00
SS-50 Backplanes 4,6,8,12,&16 position @ 15.00 per slot w/o connectors	
SS-30 8 pos. BP w/o connect.	$ 39.00
Transition Card $49.00, A/T	$ 95.00
Molex Gold Male $1.60, Female	$ 1.60
Molex Tin Male $ .40, Female	$ .50

'68' MICRO JOURNAL ADVERTISERS INDEX

This Index is provided as a reader service. The publisher does not assume any liability for omissions or errors.

THE CHIEFTAIN™ 5¼-INCH WINCHESTER HARD DISK COMPUTER

SO ADVANCED IN SO MANY WAYS . . . AND SO COST-EFFECTIVE . . . IT OBSOLETES MOST OTHER SYSTEMS AVAILABLE TODAY AT ANY PRICE.

● HARD DISK SYSTEM CAPACITY

The Chieftain series includes 5¼- and 8-inch Winchesters that range from 4- to 60-megabyte capacity, and higher as technology advances. All hard disk Chieftains include 64-k memory with two serial ports and DOS69D disk operating system.

● LIGHTNING ACCESS TIME

Average access time for 5¼-inch Winchesters is 70-msec, comparable to far more costly hard disk systems. That means data transfer *ten-times faster* than floppy disk systems.

● 2-MHZ OPERATION

All Chieftains operate at 2-MHz, regardless of disk storage type or operating system used. Compare this to other hard disk systems, no matter *how* much they cost!

● DMA DATA TRANSFER

DMA data transfer to-and-from tape and disk is provided for optimum speed. A special design technique eliminates the necessity of halting the processor to wait for data which normally transfers at a slower speed, determined by the rotational velocity of the disk.

● RUNS UNDER DOS OR OS-9

No matter which Chieftain you select . . . 5¼- or 8-inch floppy, or 5¼- or 8-inch Winchester with tape or floppy back-up . . . they *all* run under DOS or OS-9 with *no need* to modify hardware or software.

● UNBOUNDED FLEXIBILITY

You'll probably never use it, but any Chieftain hard disk system can drive up to 20 other Winchesters, and four tape drives, with a single DMA interface board!

● SMOKE SIGNAL'S HERITAGE OF EXCELLENCE

This new-generation computer is accompanied by the same *Endurance-Certified* quality Dealers and end-users all over the world have come to expect from Smoke Signal. And support, software selection and extremely competitive pricing are very much a part of that enviable reputation.

20-Megabyte Tape Streamer Back-Up Option

Available with all Chieftain hard disk configurations. This cartridge tape capability provides full 20-megabyte disk back-up in less than five minutes with just one command, or copy command for individual file transfers. Transfers data tape-to-disk or disk-to-tape. Floppy back-up is also available in a variety of configurations.

The Chieftain Computer Systems:

Here are the Chieftain 6809-based hard disk computers that are destined to change the data processing industry . . .

☐ **CHIEFTAIN 95W4**
4-megabyte, 5¼-inch Winchester with a 360-k floppy disk drive (pictured).

☐ **CHIEFTAIN 95XW4**
4-megabyte, 5¼-inch Winchester with a 750-k octo-density floppy disk drive.

☐ **CHIEFTAIN 98W15**
15-megabyte, 5¼-inch Winchester with a 1-megabyte 8-inch floppy disk drive.

☐ **CHIEFTAIN 9W15T20**
15-megabyte, 5¼-inch Winchester with a 20-megabyte tape streamer.

Write or call today for details (including the remarkably low prices) on the total Chieftain Series . . . and on dealership opportunities.

SMOKE SIGNAL BROADCASTING®
31336 VIA COLINAS
WESTLAKE VILLAGE, CA 91362
TEL (213) 889-9340

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Telephone ( ) _____

Intelligent Serial I/O Processor Board Now Available

The GIMIX Intelligent Three-port RS-232C Serial I/O Interface can significantly increase throughput of a multi-user system by reducing the number of interrupts between user terminals and the host CPU. The Intelligent I/O Board accomplishes this by buffering data transfers between system and users and preprocessing of the data.

Appropriate on-board software and operating system drivers are required. Software and drivers for OS-9 Level 2 will be available shortly from GIMIX.

- INDEPENDENT ON-BOARD **2MHZ 68B09** CPU
- UP TO **20K** OF ON-BOARD MEMORY (EPROM and RAM)
- BUFFERED DATA TRANSFER BETWEEN HOST AND ON-BOARD CPU USING A **Z8038** FIO I/O INTERFACE UNIT
- THREE **RS-232C** SERIAL I/O PORTS **(6551As)** WITH SOFTWARE SELECTABLE BAUD RATES, WORD LENGTH, STOP BITS, PARITY

Standard Version Including 4K RAM (Without Software) $438.11

— ***PARALLEL VERSION COMING SOON*** —

Uniflex For GIMIX Winchester Systems

TSC will be providing UniFLEX compatible with GIMIX Winchester systems. The NEW versions of UniFLEX for use with the Winchester systems will be delivered on 5" media as well as 8" media.

GIMIX 30 Pin Prototyping Board Now Available

- Double sided with plated thru holes and gridded power and ground lines.
- 8 rows of pads on .100 x .300 centers: up to 41 fourteen pin ICs.
- Accepts standard 6, 8, 14, 16, 20, 24, and 40 pin DIP devices.
- The entire top edge has pads for .100 x .100 header (ribbon) connectors.
- Pads for solder connections or .100 center headers on all 30 bus lines.
- Accepts 3 TO-220 regulators, 1 on the +8V & 1 ea. on the +/– 16V lines.
- Provisions for decoupling caps distributed throughout the array.
- Can be used with wire wrap, wiring pencil, solder wiring, etc.

With gold bus connectors and heat sinks (unassembled) $38.33

Now Available From GIMIX

(U.S. & Canada Only)

THE WINDRUSH EPROM PROGRAMMER

- ★ Probably the most versatile EPROM PROGRAMMER available. Interface & software for EXORcisor - II (fully addressable) and S-50 bus systems.
- ★ PROGRAMS AND VERIFIES 2508/2708, 2516/2716 (SINGLE AND TRI-VOLT TYPES) 2532, 2732, 2732A, 2564, 2764 and the 128K TMS2528 (16K x 8) - - - - - - - WITHOUT ADDITIONAL 'PERSONALITY' MODULES - - - - - - -.
- ★ PROGRAMMER extends out to your work area via 5' of twisted pair cable.
- ★ EXTENSIVE COMMANDS MENU... MOVE DATE, READ, PROGRAM, VERIFY EPROMS, EXAMINE/CHANGE BUFFER, FORMATTED DUMP OF BUFFER, FILL BUFFER.
- ★ Fully documented user's manual w/schematics & theory of operation. Professionally finished PCBs w/solder resist & component overlay.
- ★ SOFTWARE AVAILABLE FOR FLEX 2/9, SSB, OS-9 (LVL 1 NOW, LVL 2 LATER) and MDOS... All source files supplied. Specify disk size please!

NOTE: One version is supplied FREE. Extra versions: $25.00 each.

S-30 Interface/Programmer/Baseplate/Cable $375.00
EXORcisor Interface/Programmer/Baseplate/Cable $395.00

1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510
TWX 910-221-4055

GIMIX INC.

The Original FLEX for Color Computers

- Upgrade to 64K
- RS to FLEX, FLEX to RS file transfer ability
- Create your own character set
- Automatic recognition of single or double density and single or doubled sided
- All features available for either single or multiple drive systems
- Settable Disk Drive Seek Rates
- Faster High Resolution Video Display with 5 different formats
- Save RS Basic from RAM to Disk
- Move RS Basic to RAM
- Load and save function on FLEX disk
- 13 Support Commands & with Source Text

Languages Available

Pascal, Fortran, RS Basic, RS Assembler, TSC Basic, TSC Assembler, Relocating Assembler, Macro Assembler, Mumps

If you are tired of playing games on your TRS-80C™ Color Computer, or find that you are handicapped by the limitations of the RS BASIC in trying to write a Program that will allow you to actually USE the Color Computer as a COMPUTER, YOU ARE READY TO MOVE UP TO THE **FLEX9**™ Operating System. If you want to have REAL PROGRAMMING POWER, using an Extremely Powerful Business BASIC, PASCALs, C Compilers, a full-blown Macro Assembler with a Library capability so you are not continuously "reinventing the wheel", YOU ARE READY TO MOVE UP TO THE **FLEX9**™ Operating System. If you would like to see if YOU REALLY COULD USE A COMPUTER IN YOUR BUSINESS, or begin to make your Computer start PAYING IT'S OWN WAY by doing some Computer Work for the millions of small businesses around you, such as Wordprocessing, Payroll, Accounting, Inventory, etc., then YOU ARE READY TO MOVE UP TO THE **FLEX9**™ Operating System. How?? **DATA-COMP** has the way!

DATA-COMP's FLEX9™ Conversion for the TRS-80C™ Color Computer was designed for the SERIOUS COMPUTER **USER:** with features like greatly increased Display Screens, WITH Lower Case Letters, so you can put a FULL Menu on ONE Screen, or see SEVERAL Paragraphs at the same time; with features like providing a FULL Keyboard so you have FULL Control of your Computer AND it's Programs NATURALLY, without needing a chart to see what Key Combination will give you what function; with USER ORIENTED functions to make using the Operating System natural, like having the Computer AUTOMATICALLY determine what type of Disk is being used in what type of Disk Drive and working accordingly, rather that you have to specify each and every thing for it, or like having the Computer work with the Printer you have been using all along without you having to tell the new Operating System what is there, etc., etc., etc.

DATA-COMP has everything you need to make your TRS-80C™ Color Computer WORK for YOU: from Parts and Pieces to Full, Ready To Use SYSTEMS. DATA-COMP designs, sells, services, and SUPPORTS Computer SYSTEMS, not just Software. CALL DATA-COMP TODAY to make your Computer WORK FOR YOU!

System Requirements

FLEX9 Special General Version x Editor & Assembler (which normally sell for $50.00 ea.)	$150.00
F-MATE(RS) FLEX9 Conversion Rout. for the RS Disk Controller when purchased with Special General FLEX9 Sys.	$69.95
when purchased without the General FLEX9 Sys.	$79.95
Set of Eight 6 K RAM Chips w Mod. Instructions	$69.96
Color Computer with 64K RAM and EXT. BASIC	$499.95
Color Computer with 16K RAM	$299.95
Color Computer with 16K RAM and EXT. BASIC	$389.95

SPECIAL SYSTEM PACKAGES

6 K Radio Shack COLOR COMPUTER, Radio Shack COLOR D SK CONTROLLER, a Disk Drive System, Special General Version of FLEX9™, F-MATE(RS)™ and a Box of 10 Double Density Diskettes; a COMPLETE, ready to run SYSTEM on your Color TV Set. $1249.95

DISK DRIVE PACKAGES, etc.

These Packages include the Radio Shack Disk Controller, Disk Drives with Power Supply and Cabinet, and Disk Drive Cable:

PAK #1 — 1 Single Sided, Double Density Sys.	$499.95
PAK #2 — 2 Single Sided, Double Density Sys.	$769.95
PAK #3 — 1 Double Sided, Double Density Sys.	$599.95
PAK #4 — 2 Double Sided, Double Density Sys.	$949.95
PAK #5 — 2 Qume Thinline Double Sided Double Density Sys.	$784.95

PARTS AND PIECES

Radio Shack Disk Controller	$179.95
1 Tandon Single Sided, Double Density Disk Drive	$249.95
1 Tandon Double Sided, Double Density Disk Drive	$349.95
1 Qume Thinline Double Sided, Double Density	$279.95
Single Drive Cabinet with Power Supply	$89.95
Double Drive Cabinet with Power Supply	$109.95
Single Drive Disk Cable for RS Controller	$24.95
Double Drive Disk Cable for RS Controller	$34.95
Micro Tech rods., Inc. **LOWER CASE ROM** Adapter	$74.95
Radio Shack **BASIC Version 1.1 ROM**	$34.95
Radio Shack **Extended Basic ROM**	$89.95

SOFTWARE

CHESS 6809

Requires FLEX™ and one of the following CRT terminals

Now Runs On Any Type Terminal

Features:

- **Two display boards.**
- **Four levels of play.**
- **Point scoring system.**
- **Play white or black.**
- **Change or set-up boards/piece positions.**
- **Forfeit move.**
- **Swap sides.**
- **Make move and swap sides.**
- **Change skill level.**
- **Stop and restart game.**
- **Solve 'Mate in 1-2-3-4' moves.**

$79.95 Specify 5" or 8" disk

This is one of the strongest CHESS programs running on any microcomputer, estimated USCF Rating 1600 + .

Note: Personal checks allow 3-4 week delivery.

DIET-TRAC Forecaster

A Diet Planning and Analysis Program

DIET-TRAC Forecaster is a program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P F %) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual.

Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individuals are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. When a weight goal is given (either gain or loss), and a calorie plan is agreed upon between the computer and the individual, the number of days to reach the weight goal is projected. The starting and ending rate of weight loss is calculated, and a daily calendar with each day's predicted weight for a 30-day period is printed.

FLEX VERSION	**$59.95**
UniFLEX VERSION	**$89.95**

PRINTERS

The Epson MX-80
$495.00
The Epson MX-100
$725.00

MX-70 $355.00 MX-80 FT $575.00

☆

Color Computer External Terminal Program
$19.95

☆

Color Computer Stylograph Running Under F-Mate
$195.00

MasterCard VISA

DATA-COMP SOUTH EAST MEDIA

P.O. Box 794 *HIXSON, TN 37343*

1-615-842-4601

Verbatim Diskettes.

5" Soft Sector Disks

Single Side Single Density	**$2.75 ea.**
Single Side Double Density	**$2.75 ea.**
Double Side Double Density	**$4.92 ea.**
Plastic Storage Box	**$2.00 ea.**

8" Soft Sector Disks

Single Side Single Density	**$3.75 ea.**
Single Side Double Density	**$4.10 ea.**
Double Side Double Density	**$4.75 ea.**
Plastic Library Box	**$5.00 ea.**

Foreign Orders Add 10% Surface—20% Air Mail

WINCHESTER BACKUP UTILITIES

The following utilities allow the backup of any size disk system to any size diskettes.

By simply inserting diskettes as requested by COPYMULT, a large disk system (Winchester, etc.) may be downloaded to your present floppy disk system, any size. No need to fiddle with directory deletions or any of the other tedious operations that must be done using a normal copy routine.

COPYMULT-CMD understands normal "copy" syntax and always keeps up with files already copied by maintaining directories for both host and receiving disk system, thus eliminating hours of tedious keyboard entries and other time consuming cleanup chores.

BACKUP-CMD is a special program that downloads "random" type files, any size.

RESTORE-CMD a special program to restructure copied "random" files for copying, or recopying back to the host system.

FREELINK-CMDa "bonus" utility that "relinks" the free chain of a floppy or hard disk thereby eliminating fragmentation.

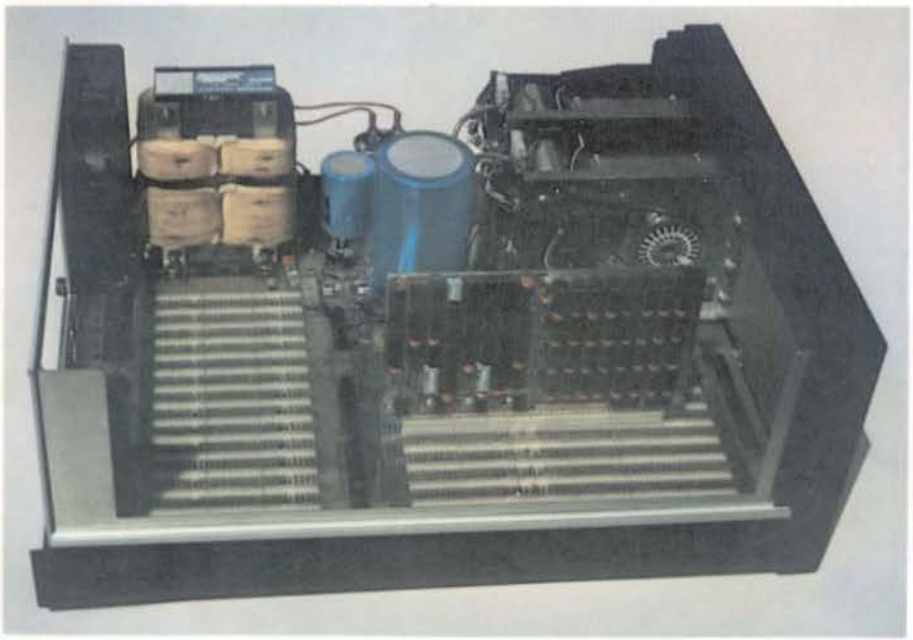
** Completely documented source files included.
** ALL 3 Programs $99.50 on 8" diskette

Data-Comp—South East Media & 68 Micro Journal Are Divisions of CPI

68 Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

Second Class Postage Paid
At Chattanooga, TN
ISSN 0194-5025

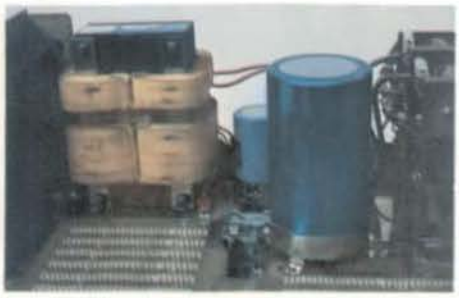
HELIX™



THE MAINFRAME
- Industry Standard Optima™ Cabinet
- Largest Constant Voltage Power Supply in the industry
- S-64 Bus gives 16 Bit Power and S-50 Bus Compatibility
- 10 Main (S-84) Slots
- 14 I/O (S-30) Slots plus 2 On-board
- On-board Baud Rate Generator to 38.4Kb
- Space and Power for two 5¼" Disk Drives
- Full Address Decoding for I/O Slots
- Two RS-232 Serial and Two parallel Ports On-board
- Single Board Construction for Reliability
- Faraday Shielded Bus Lines give "Text Book Clean" Signals

THE PROCESSORS
6809
- Standard 2 MHz Operation
- Standard DAT Compatible with GIMIX and SWTPC
- Standard 6840 Interval Timer
- Standard 1K Scratchpad RAM
- Standard Clock/Calendar with Battery
- Provision for Programmers Console

68000
- Standard 8 MHz Operation
- Memory Management Hardware
- Provision for Programmers Console
- 16 Bit Power and 8 Bit Compatibility



The HELIX™ computer system represents the latest advance in S-50 bus computer systems. Relying on the physical nature of S-50 bus connectors to guarantee compatibility, the HELIX adds 14 bus lines (becoming S-64) to allow a 68000 processor to operate with full 16 bit data transfer and 24 bit addressing, while at the same time providing full interchangability with existing S-50 components.

Offered with a selection of processors, memories, and peripheral controllers, a HELIX system can be configured for applications ranging from advanced hobbyist to multiterminal time-sharing.

Designed to offer the utmost in speed, reliability, and utility at a reasonable price, it represents a new standard of quality for those who require a professionally designed computer for professional use.

THE POWER SUPPLY
- Ferro-resonant Transformer for Line Noise and Under-Voltage Protection
- Conservative 25 Amps at 8.5 Volts
- Conservative 5 Amps at ±16 Volts
- Conservative Component Rating for Reliability

THE COMPONENTS
- Fully Socketed
- Gold Plated Bus Connectors
- Only "B" Series 68XX Components Used
- Only Top Grade Logic Circuits Used
- Industrial Grade Components Throughout

THE MEMORIES
DM-64
- Field Proven
- Proprietary Memory Control Logic
- Fully Transparent Refresh
- Tested at 2.5 MHz Operation

DM-512
- 512K Bytes on a Single S-64 Board
- 18 Bit Power and 8 Bit Compatibility
- Runs in Existing S-50 Systems where Physical Space Allows
- Full 24 Bit Addressing
- Fully Transparent Refresh

THE PRICES
Because of the variety of configurations possible, full pricing cannot be given. Representative prices are:

- 64K 6809 HELIX $1995
- 64K 68000 HELIX $2595
- 512K 6809 HELIX $4450
- 512K 68000 HELIX $4995

HAZELWOOD COMPUTER SYSTEMS

907 E. Terra, O'Fallon, Missouri 63366 (314) 281-1055

Dealer and OEM Inquiries Invited. We support our Dealers.

Optima is a Trademark of Scientific-Atlanta